TRABAJO DE FIN DE GRADO

Grado en Matemáticas

Título: Modelos Lineales Dinámicos en Sensórica: Desarrollo de un paquete en R

Autor: Pan Ye

Director: Josep Anton Sànchez Espigares

Departamento: Estadística e Investigación Operativa

Convocatoria: 2020

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Modelos Lineales Dinámicos en Sensórica: Desarrollo de un paquete en R

Trabajo Final de Máster

Máster en Estadística e Investigación Operativa

La Facultad de Matemáticas y Estadística

Universidad Politécnica de Cataluña

Pan Ye

Director: Josep Anton Sànchez Espigares

Junio 2020

Agradecimientos

Ante todo, deseo expresar mi sincera gratitud al director de esta tesis de Máster, Josep Anton Sànchez Espigares, por la dedicación y apoyo que ha brindado a este trabajo, por el respeto a mis ideas y por la dirección y el rigor ha facilitado a las mismas.

Asimismo, agradezco al MESIO que me ha dado esta oportunidad de consolidar mis conocimientos en el campo estadístico con el fin de realizar este trabajo. Y además, doy las gracias a los compañeros y profesores que me han acompañado en esta etapa de mi vida.

Finalmente, presento mis agradecimientos al personal médico y todas las personas que se han dedicado en el período de Covid-19.

RESUMEN

El concepto de sensórica es diseñar dispositivos que registran señales crudas. Sin embargo, la señal que a veces resulta interés no es la que estamos registrando y es necesario de establecer alguno tipo de modelo que relacione las señales crudas con la de interés. En esta tesis, proponemos los modelos lineales dinámicos (DLMs), porque las señales son series temporales interrelacionadas donde algunas de ellas no se observan directamente. Este tipo de modelos plantea no solo la evolución de las series latentes, sino también la relación de estas series con las observadas para tener una estimación mediante el método de Kalman Filter.

En este tabajo construimos un paquete en R que incluye la metodología anterior y que a partir de unas señales crudas indicando qué modelo dinámico lineal relaciona esas señales con las de interés. Este paquete, en una situación real, está implementado dentro de sensor para reportar directamente los datos transformados en lugar de los valores registrados. El objetivo de esta memoria es introducir los modelos dinámicos lineales y los conceptos básicos relacionado con la construcción de un paquete, describir las clases y los métodos definidos en el sistema orientado a objetos `S4` y , finalmente, ilustrar el uso de este paquete con un ejemplo concreto, el análisis de la salinidad en terreno.

Palabras claves: modelos dinámicos lineales, filtro de Kalman, redes neuronales recurrentes, conductividad eléctrica, librería de R.

MSC2000: 37M10, 62M05, 82-04.

Abstract

The concept of sensorial is to design devices that record raw signals. However, sometime the signal of interest is not the one we are registering and it is necessary to establish some kind of model that relates the raw signals with the signal of interest. In this thesis, we propose dynamic linear models (DLMs), because the signals are interrelated time series where some of them are not directly observed. This type of model proposes not only the evolution of the latent series, but also the relationship of these series with those observed to have an estimate using the Kalman Filter method.

In this work we build a package in *R* that includes the previous methodology, based on raw signals indicating which linear dynamic model relates those signals to those of interest. This package, in a real situation, is implemented within the sensor to directly report the transformed data instead of the recorded values. The purpose of this report is to introduce linear dynamic models and basic concepts related to package construction, to describe the classes and methods defined in the `S4` object-oriented system, and finally to illustrate the use of this package with a concrete example, the analysis of salinity in the crop land

Key words: dynamic linear models, Kalman filter, recurrent neural network , electrical conductivity, R packages.

MSC2000: 37M10, 62M05, 82-04.

Índice general

Índice de figuras

Introducción

La sanilidad es un indicador importante para la productividad del cultivo. En el presente estudio, los dispositivos sensóricos operan bajo el modelo de Hilhorst, un modelo determinístico que mide la conductividad eléctrica de agua basando en unas mediciones junto con un offset predeterminado. Según los estudios previos, la conductividad eléctrica de agua es una manera consistente de medir la salinidad en un terreno cultivo. Sin embargo, el uso de un valor general de offset es cuestionable y muchas investigaciones se utilizan un valor distinto.

A partir del método de Hilhorst, Basem Aljoumani aplica un enfoque estadístico en el mismo problema en su tesis doctorado considerando que la conductividad eléctrica de agua y el offset son dos variables desconocidas en el sistema. Mediante unas transformaciones simples del modelo de Hilhorst, el mismo problema se puede formular por los modelos dinámicos lineales. Basando el análisis de Basem et al. [2018], el objetivo principal de esta tesis es crear un paquete en R para estandarizar el proceso de análisis y simplificar su uso.

Para entender este enfoque, el estudio de los modelos dinámicos lineales es necesario y el núcleo es estos modelos es un método recursivo de estimadores diseñado por Kalman que tiene nombre el filtro de Kalman. No obstante, el paquete `sm4sd` llega más allá de este análisis, su objetivo es dar una interfaz de multiples modelos posibles para el problema de salinidad y estos modelos distintos se forman clases distintas pero conservan la relación jerarquica.

Para indicar esta relación jerarquica, el uso del sitema `S4` se considera en la definición del paquete. `S4` es un sistema orientado a objeto que es más formal y rigor respecto a `S3`. Por esta formalidad, el sistema `S4` hereda unas propiedades de la programación orientada a objetos que son el encapsulamiento, la heriencia y el polimorfimos. A parte de los modelos, `sm4sd` incluye las redes neuronales recurrentes con el fin de realizar simulaciones basando los patrones capturados.

En resumen, esta tesis está compuesta por dos bloques, la construcción del paquete `sm4sd` y la elaboración de la memoria. El tiempo de indicación para cada parte es 50 %. En concreto, esta memoría se estructura en los capítulos:

1. Introducción del trabajo;
2. Marco teóricos donde concentramos el estudio de los modelos dinámicos lineales y las redes neuronales recurrente;
3. Introducción de los conceptos de R, los componentes de un paquete, el sistema `S4` y las clases y los métodos definidos en `sm4sd`;
4. Aplicación de `sm4sd`, en este capítulo ilustraremos el uso de este paquete ba-

sando un conjunto de datos de laboratorio;
5. Conclusión de la tesis, una pequeña reflexión del trabajo y las posibles mejoras de `sm4sd` en la próxima versión;
6. Referencias usadas y dos apéndices contienen los códigos de los métodos y el manual de ayuda del paquete.

Sobres las herramientas usadas, destaco un par de ellos:

1. la libreria `bookdown` en R, una herramienta para elaborar los ebook. En este caso, esta tesis se elabora por `bookdown` en `Rstudio`;
2. la libreria `roxygen2`, una libraria facilita la elaboración de las ayudas de paquete;
3. la libreria `neuralnetwork` de *Latex*, el uso de esta librería permite realizar figuras de las redes neuronales como la figura 1.6;
4. y, finalmente, la librería `dlm` que porporciona la definición de los modelos dinámicos lineales y las herramientas necesarias.

Capítulo 1

DLM y RNN

Este capítulo consiste en el estudio de dos modelos aplicados en la librería `sm4sd`, el modelo dinámico lineal y el modelo de las redes neuronales recurrentes, que está dividido en dos secciones distintas.

1. En la primera sección haremos una revisión de los modelos dinámicos lineales, una exploración de la distribución de los distintos estimadores y la aplicación del método de Kalman, la validación del modelo basando el proceso de innovación y, finalmente, una aplicación real.
2. La segunda sección se enfoca en un breve estudio de las Redes Neuronales Recurrentes entrenadas por el filtro de Kalman extendida al caso no lineal.

1.1. Modelo Lineal Dinámico

En la estadística, más concreto el caso de regresión, el modelo lineal o la regresión lineal es un modelo matemático usado para aproximar la relación entre la variable explicativa X (factores y covariantes) y la variable respuesta Y bajo las siguientes hipótesis:

1. la relación de dependiencia es lineal denotada por el vector β, un vector de coeficientes constates que explicita dicha relación en una forma aditiva entre los componentes X y β.
2. Conociendo el valor de X, Y sigue una distribución normal y, además, si $Y_1, \cdots, Y_n$ realizaciones de Y con las varianzas $\sigma_1^2, \cdots, \sigma_n^2$, entonces se verifica $\forall i \neq j$, $cov(\sigma_i^2, \sigma_j^2) = 0$ y $\forall i, \sigma_i = \sigma$.

En una lengua más matemática, el modelo lineal (LM: *Linear Model* en inglés) se puede formula como siguiente: dados $n \in \mathbb{N}$, $X_t \in \mathbb{R}^p$, $Y_t \in \mathbb{R}$ y $\beta \in \mathbb{R}^{p+1}$ tal que Y_t es una función en X_t y β^t:

$$Y_t = \beta^t * \begin{pmatrix} 1 \\ X_t \end{pmatrix} + \epsilon_t, \quad \forall t \in 1, \cdots, n$$

donde el coeficiente 1 está relacionado con el término *intercept*[1] y ϵ_t es el error independiente a la variable explicativa y $\forall t, \epsilon_t \sim N(0, \sigma^2)$ y $\forall i \neq j, cov(\sigma_i, \sigma_j) = 0$.

El LM tiene un uso muy amplio, sin embargo, vivimos en un mundo tan diverso donde se encuentran fácilmente restricciones y limitaciones de su aplicación. Para enfrontar estos problemas, el modelo lineal generalisado (GLM)[2] fue introducido por Nelder y Wedderburn en el año 1972 que es una genralización del LM incluyendo una función de link[3] y otros modelos de la familia exponencial para el residuo como Poisson, Binomial y etc.

En estos dos casos, el vector de coeficientes β se mantiene constante. Sin embargo, en el análisis de las series temporales, un se puede encontrar fácilmente que estos coeficientes varian respecto al tiempo t y se denota por β_t. Manteniendo el restos de las hipótesis de LM, nos lleva una nueva variante cuyo nombre Modelos Lineales Dinámicos (DLM: *Dynamic Linear Model*), que es un caso particular de los modelos de espacio y estado (SSM: *State Space Model*). El SSM o *state-space representation* es un model basado en el sistema dinámico. En la ingenería de control, se expresan los fenómenos mediante unas ecuaciones diferenciales de primer orden conocidas como ecuaciones de evolución o estado, $g(\theta_t)$, y una forma simplificada de representa este tipo de modelo es[4]:

$$\begin{aligned} Y_t &= f(\theta_t) + v_t, \\ \theta_t &= g(\theta_{t-1}) + w_t. \end{aligned} \tag{1.1}$$

Si $Y_t \in \mathbb{R}^n$ y $\theta_t \in \mathbb{R}^p$, tenemos f y g aplicaciones definidas en los ternos $(\mathbb{R}^p, \mathbb{R}^n)$ y $(\mathbb{R}^p, \mathbb{R}^p)$ respectivamente. Y si además restrigimos f y g al espacio de aplicaciones lineales y denotamos F_t y G_t matrices asociadas a las aplicaciónes f y g en el instante t, la ecuación 1.1 se puede reescribir en la siguiente forma:

$$\begin{aligned} Y_t &= F_t \cdot \theta_t + v_t \qquad V_t \sim N_n(0, \Sigma_v) \\ \theta_t &= G_t \cdot \theta_{t-1} + w_t \quad w_t \sim N_p(0, \Sigma_w) \end{aligned}$$

y se correspon al DLM.

Una de las caracteristicas que hace los seres humanos especiales es anhelar saber el origen de todo y ¿cuál fue el origen de la idea DLM? Según las literaturas, tendremos que rastrear de nuevo al año 1960. En aquell año, R.E. Kalman presentó un nuevo *approach* en Kalman [1960] relacionado con el filtrado lineal y el problema de predicción como una genralización del filtro de Wienner para las series temporales no estacionarias basando la representación de Bode-Shannon. El filtro de Wienner, en el

[1] En inglés, donde indica el caso de que el X_t toma el valor nulo.

[2] GLM también es una optativa del Máster en Estadística e Investigación Operativa, personalmente, recomendo mucho esta asginatura por su uso amplio en el mundo analítico. Para lectores que ansiar saber más detalles, un bueno práctico es consultar la wikipedia.

[3] una función de link representa la relación entre la respueta y el predictor lineal $\beta^t * X$. En este caso, la variable X incluye el termino intercept.

[4] En esta representación, se ignora la variable de control, un elemento clave en la ingenería de control.

principio, solo operó sobre el pasado y el presente de las series temporales (interpolación y alisado), y el problema de previsión (en caso continuo) no estaba disponible hasta la segunda guerra mundial. Sin embargo, el dicho problema en las series temporales continuas está sujetado la resolución de la ecuación integral de Wiener-Hopf que tiene limitaciónes teóricas y prácticas.

Kalman describió unos estimadores óptimos en su artículo para estos tres problemas mediantes la teoría de la probabilidad y estadística con una extra demostración basada en la teoría de geometria, más concreto la proyección ortogonal. Considerando el problema $Y(t) = X(t) + \epsilon(t)$[5] donde

1. Y: fenómenos observados;
2. X: la variable aleatoria que representan a la señal real oculta;
3. ϵ: la variable del ruido;
4. t, la variable temporal.

Uno dels conceptos más frecuentes hoy en día es que $Y(t)$ sea una nueva variable aleatoria por ser la suma de las dos. Dando s un índice del tiempo, se puede definir los siguientes problemas: alisado (*data-smoothing*: $s < t$), filtrado (*filtering*: $s = t$) y predicción (*forecasting*: $s > t$). Dado un conjunto de valores medidos de $Y(t)$, $\{Y(t_0) = y_{t_0}, \cdots, Y(t_n) = y_{t_n}\}$, entonces la distribución de probabilidad condicionada de $X(s)$ en el punto x es

$$P(X(s) < x | Y(t_0) = y_{t_0}, \cdots, Y(t_n) = y_{t_n}) = F(x)$$

y denota $X_{s|t}$ el estimador estadístico de $X(s)$ con las informaciones de $\{Y(t)\} = \{Y(t_0), \cdots, Y(t_n)\}$.

Es obvio que el estimador $X_{s|t}$ es una nueva r.v. por ser una función de $\{Y(t)\}$. Por tanto, para determinar $X_{s|t}$ consideramos una función de pérdida (*Loss Function*) L positiva, no decreciente y simétrica respecto al punto 0 con $L(0) = 0$ del error de estimación $e = X(s) - X_{s|t}$. Bajo la hipótesis de la distribución condicionada y la función de pérdida, el mismo artículo afirma que el estimador óptimo $X^*_{s|t}$ que minimiza la pérdidad media $\mathbb{E}[L(e)] = \mathbb{E}\left[\mathbb{E}[L(e)|\{Y_t\}]\right]$ es

$$X^*_{s|t} = E[X(s)|\{Y_t\}].$$

Parecen unos resultados muy evidentes en el presente. Si $X(t)$ y $\epsilon(t)$ son procesos guasianos, entonces el resultado anterior siempre es cierto. Para caso multivariante, el mismo resultado se puede derivar por la teoria de proyección ortogonal (consulta la sección *Orthgonal Projections* de Kalman 1960).

En el resto de esta sección exploraremos los siguientes puntos:

1. el formalismo del modelo lineal dinámico y algunos ejemplos ilustrativos;
2. el filtro de Kalman con los detalles técnicos relacionado con los problemas alisado, filtrado y predicción;
3. el proceso de inovación y la validación del modelo;
4. y la aplicación del caso de estudio, análisis de la conductividad eléctrica del agua en los terrenos.

[5] Las notaciones del problema han sido unificadas.

1.1.1. Formalismo y ejemplos

A lo largo de este trabajo, usaremos la defición y las notaciones del libro *Bayesian Forecasting and Dynamic Models* de West & Harrison.

El modelo lineal dinámico (DLM) está caraterizado por un conjunto cuaternario

$$\{F, G, V, W\}_t = \{F_t, G_t, V_t, W_t\}$$

para cada t define un modelo secuencial relacionando Y_t a un vector de parámetros θ_t en cada instante mediante las ecuaciones siguientes:

$$Y_t = F_t\theta_t + v_t, \quad v_t \sim N(0, V_t), \tag{1.2}$$

y

$$\theta_t = G_t\theta_{t-1} + w_t, \quad w_t \sim N(0, W_t). \tag{1.3}$$

Las $\{v_t\}_t$ y $\{w_t\}_t$ definen dos secuencias de variables aleatorias guasianas e independientes y para cada t, $\forall i, j, cov(v_t^{(i)}, w_t^{(j)}) = 0$. La ecuación 1.2 describe la distribución condicionada de las observaciones Y_t en θ_t y donde la ecuación 1.3 es la evolución de θ_t especto a t. En este caso, la evolución es un proceso markoviano porque θ_t solo depende del instante anterior. La segunda ecuación se conoce como la ecuación de estado del sistema.

Para resumir, un DLM es un sistema de ecuaciones lineales y guasianas con el proceso de estado markoviano y las observaciones condicionandas al estado independientes. Para cada t, tenemos los siguientes elementos:

a) $Y_t \in \mathbb{R}^{p\times 1}$ es el vector de variables de observación que representa las mediciones no exactas de un sistema ;
b) $\theta_t \in \mathbb{R}^{n\times 1}$ es el vector de variables latentes conocidas como estado y está relacionada con las variables independientes en un sistema;
c) $F_t \in \mathbb{R}^{p\times n}$ es la matriz de diseño de los valores conocidos de las variables independientes;
d) $G_t \in \mathbb{R}^{n\times n}$ es la matriz de evolución que describe la transición del estado de instante $t-1$ a t;
e) v_t y w_t son errores de la observación y del estado con media 0 y matriz de varianza $V_t \in \mathbb{R}^{p\times p}$ y $W_t \in \mathbb{R}^{n\times n}$ respectivamente;
f) y, finalmente, $\mu_t = \mathbb{E}[Y_t|\theta_t] = F_t\theta_t$ es el nivel de la observación.

En general, el valor de n es menor que p y, visualmente, la estructura de depedencia del modelo se puede representarse en el siguiente diagrama:

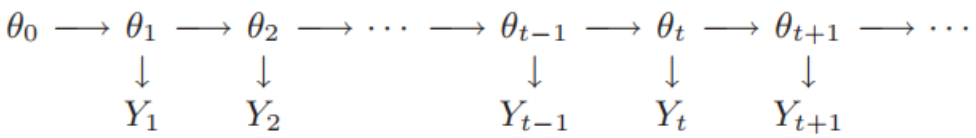


Figura 1.1: Estructura de dependencia de DLM

Dependiento la propiedad del cuaterno anterior, podemos obtener los siguientes subclases de DLM por intereres específicos:

i) Si el par $\{F, G\}_t$ es constante para todos los t, entonces el modelo se refiere como un DLM de series temporales o TSDLM.
ii) Decimos que Un TSDLM es un DLM constante si la varianza de observación y evolución es constantes.

Además, si $\{Y_t\}_t$ son escalares, en el caso $p = 1$, estamos hablando un DLM univariantes. Referimos a partir de ahora DLM como los modelos dinámicos lineales univariantes. El segundo tipo de subclase tiene una importancia enorme ya que incluye todos los modelos lineales clásicos, ARIMA. Está transformación bidirección no está incluida en este trabajo, si uno le interesa, puede consultar el capítulo 3.2.5 *DLM representation of ARIMA models* del P.Govanni et al. [2009]. Sin embargo, ilustraremos una transformación de AR(p) en el primer ejemplo.

Modelo de Paseo aleatorio y AR(p) El modelo más simple para series temporales univariantes es el paseo aleatorio con ruido blanco, definido por

$$\begin{aligned} Y_t &= \mu_t + v_t, \qquad v_t \sim N(0, V) \\ \mu_t &= \mu_{t-1} + w_t, \quad w_t \sim N(0, W), \end{aligned} \tag{1.4}$$

donde las secuencias $\{v_t\}$ y $\{w_t\}$ son independientes internamente y mutualmente. Este es un modelo con $p = n = 1$, $\theta_t = \mu_t$ con F_t y G_t constantes e igual a la unidad. Si hacemos las transformaciones necesarias, este modelo se corresponde a AR(1) con el coeficiente 1. Sustituiendo μ_t de la ecuación de observación por la de evolución, tenemos lo siguiente

$$Y_t = \mu_{t-1} + w_t + v_t, \quad w_t \sim N(0, W) \text{ y } v_t \sim N(0, V). \tag{1.5}$$

Expresamos μ_{t-1} en función de Y_{t-1} por 1.4

$$Y_{t-1} = \mu_{t-1} + v_{t-1} \Rightarrow \mu_{t-1} = Y_{t-1} - v_{t-1}. \tag{1.6}$$

Replazamos μ_{t-1} de 1.5 por 1.6

$$Y_t = Y_{t-1} + v_t - v_{t-1} + w_t = Y_{t-1} + w_t, \quad w_t \sim N(0, W)$$

donde $v_t - v_{t-1}$ es r.v. 0 por ser i.i.d. la secuencia $\{v_t\}$.

En el análisis clásico, los modelos autorregresivos es una subclase de los ARMA (*AutoRe-gressive Moving Average models*, también conocidos como modelos de Box-Jenkins en el caso de series estacionarios) que están caracterizados por un parámetro de retraso (o *lag*), p, y se denota por AR(p). Un AR(p) se puede escribirse como

$$Y_t = \sum_{j=1}^{p} \phi_j B^j(Y_t) + v_t,$$

donde $B^j(Y_t) = Y_{t-j}$. En general, para obtener la representación DLM de AR(p) no es tan obvia debido a las hipótesis.

Dando un AR(2), $Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + v_t$, para complir la hipótesis de independencia condicionada, las informaciones históricas Y_{t-1} y Y_{t-2} deben ir un de los componentes del estado. Si definimos la ecuación de observación como $Y_t = \theta_t$, usando la dependencia de AR(2) podemos obtener la siguiente representación del ecuación de estado

$$\begin{pmatrix} \theta_t \\ \theta_{t-1} \end{pmatrix} = \begin{pmatrix} \phi_1 & \phi_2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \theta_{t-1} \\ \theta_{t-2} \end{pmatrix} + \begin{pmatrix} v_t \\ 0 \end{pmatrix}$$

Los modelos AR(p) no tienen una representación única y una representación generalizada se peude encontrar el libro de Govanni.

Modelo de crecimiento lineal sin/con componente estacional Tanto AR(1) como su representación en DLM es un model tan básico que suele ser apropiado para muchos casos. Un modelo más sofisticado es de la serie temporal estructural. Dada una serie temporal $\{Y_t\}$, siempre se puede descomponerla de una forma aditiva en

$$Y_t = \mu_t + S_t + v_t,$$

donde μ_t indica el nivel de la serie y S_t representa el componente estacional. En caso multiplicado, $Y_t = T_t \times S_t \times v_t$, podemos obtener una descomposición aditiva en una nueva series aplicando la transformación logarítimica

$$\log Y_t = \log \mu_t + \log S_t + \log v_t.$$

Para la simplicidad, estudiamos primero el modelo sin componente estacional.

La tendencia es el componente que recoge el comportamiento o movimiento de la serie y está compuesta por dos elementos: nivel (μ_t) y pendiente (β_t). El pendiente indica la dirección de evolución de la serie en cada momento y el nivel representa la posición instantánea que está completamente determinada por la posición y su evolución anterior. Una serie temporal con la tendencia es el modelo de paseo aleatorio incluyendo el pendiente:

$$\begin{aligned} Y_t &= \mu_t + v_t, & v_t &\sim N(0, V), \\ \mu_t &= \mu_{t-1} + \beta_{t-1} + w_t^{(1)}, & w_t^{(1)} &\sim N(0, q_{11}) \\ \beta_t &= \beta_{t-1} + w_t^{(2)}, & w_t^{(2)} &\sim N(0, q_{22}), \end{aligned} \tag{1.7}$$

con los errores incorrelacionados v_t, $w_t^{(1)}$ y $w_t^{(2)}$. Este es un DLM con

$$\theta_t = \begin{pmatrix} \mu_t \\ \beta_t \end{pmatrix}, \quad G = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad W = \begin{pmatrix} q_{11} & 0 \\ 0 & q_{22} \end{pmatrix}, \quad F' = \begin{pmatrix} 1 & 0 \end{pmatrix}.$$

Las varianzas q_{11} y q_{22} se les permite ser cero.

$\{Y_t\}$, se puede pensar como la trayectoria de un fenómeno determinado y β_t de la ecuación 1.7 es la velocidad nominal instantánea al largo de tiempo. Según los valores de su varianza, se pueden describir dos tipos de trayectoria dinstintas (figura 1.2):

1. Plot 1: tendencia local donde el pendiente tiene un componente aleatorio con $q_{22} \neq 0$.
2. Plot 2: tendencia global, el pendiente es constante respecto al tiempo donde $q_{22} = 0$.

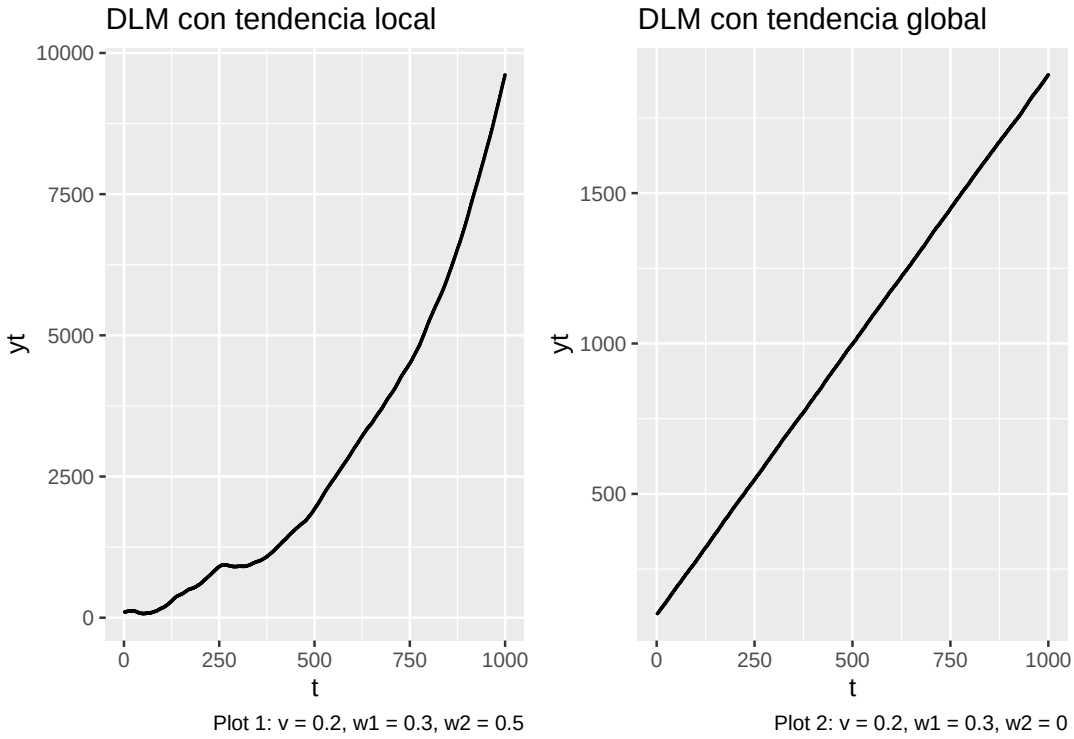


Figura 1.2: Modelos de tendencia generados por R

El componente estacional describe el movimiento oscilatorio cerca del nivel de la serie y que se repiten de forma periódica con un perodo k. En el análisis clásico de las series temporales siempre se asume este componente constante, es decir, $\sum_{j=1}^{k} B^{k-j}(S_t) = 0$. Sin embargo, en muchos casos, esta condición no es cierta menos una parte aletoria $\sum_{j=1}^{k} B^{k-j} S_t = w_t^{(3)}$ con $w_t^{(3)} \sim N(0, q_{33})$. Despejando S_t, para el caso $k = 3$, $S_t = -S_{t-1} - S_{t-2} + w_t^{(3)}$ y el modelo completo es

$$
\begin{aligned}
y_t &= \mu_t + S_t + v_t, & v_t &\sim N(0, V) \\
\mu_t &= \mu_{t-1} + \beta_{t-1} + w_t^{(1)}, & w_t^{(1)} &\sim N(0, q_{11}) \\
\beta_t &= \beta t - 1 + w_t^{(2)}, & w_t^{(2)} &\sim N(0, q_{22}), \\
S_t &= S_t = -S_{t-1} - S_{t-2} + w_t^{(3)} & w_t^{(3)} &\sim N(0, q_{33}), \\
S_{t-1} &= S_{t_1}
\end{aligned}
$$

con los coeficientes:

$$\theta_t = \begin{pmatrix} \mu_t \\ \beta_t \\ S_t \\ S_{t-1} \end{pmatrix}, \quad G = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad W = \begin{pmatrix} q_{11} & 0 & 0 & 0 \\ 0 & q_{22} & 0 & 0 \\ 0 & 0 & q_{33} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad F = \begin{pmatrix} 1 & 0 & 1 & 0 \end{pmatrix}.$$

1.1.2. Problemas S.F.P y Filtro de Kalman

Desde la aparición del filtro de Kalman (KF), numerosas aplicaciones han sido creadas como extensiones de esta técnica, por ejemplos, el sistema de navegación (Xixiang Liu, et al. [2016]), la técnica de seguimiento de objetos (Liana Taylor, et al. [2016]) y etc. Kalman Filter es un algoritmo recurrente basando en el análisis bayesiano para resolver los problmeas *smoothing*, *Filtering* y *Prediction* (S.F.P.) de DLM. En general, hallar la distribución conjunta no es una tarea fácil, incluso imposible en muchos casos, pero con las condiciones de DLM (independencia condicional) las distribuciones de estos tres problemas se pueden simplificar y convertirse en una computación recurrente donde interviene el metodo de KF.

La independencia condicional es la estructura más simple en la clase de las estructras de dependencia. Decimos que una secuencia de variables $\{y_1, \cdots, y_n\}$ son condicionalmente independientes e idénticamente distribuidas dando θ si $\pi(y_{1:n}|\theta) = \prod_{i=1}^{n} \pi(y_i|\theta)$, donde π representa la función de distribución. En este caso, se puede calcular la distribución de predicción de un paso basando todas las informaciones anteriores como

$$\begin{aligned} \pi(y_{n+1}|y_{1:n}) &= \int \pi(y_{n+1}, \theta|y_{1:n}) d\theta \\ &= \int \pi(y_{n+1}|\theta, y_{1:n}) \pi(\theta|y_{1:n}) d\theta \\ &= \int \pi(y_{n+1}|\theta) \pi(\theta|y_{1:n}) d\theta, \end{aligned}$$

donde $\pi(y_{n+1}, \theta|y_{1:n})$ es la distribución condicionada conjunta de y_{n+1} y θ, y $\pi(\theta|y_{1:n})$ representa el distribución posterior de θ en instante n. Por la formula de Bayes, la distribución posterior se puede determinar en la siguiente manera:

$$\pi(\theta|y_{1:n}) = \frac{\pi(y_{1:n}|\theta)\pi(\theta)}{\pi(y_{1:n})} \propto \prod_{i=1}^{n} \pi(y_t|\theta)\pi(\theta), \tag{1.8}$$

donde $\pi(y_{1:n})$ no depende del parámetro θ y $\pi(\theta)$ es el prior en el instante inicio.

El producto de la formula antrior nos indica para tener la distribución posterior de θ en el instante t es suficiente y necesario de computar recursivamente las distribuciones posteriores en $t = i$ y priores en $t = i + 1$ con $i = 1 : (n - 1)$. Si denotamos estas distribuciones priores como una secuencia de variables $\{\theta_t\}_t$, entonces se puede reducirse en:

$$\pi(\theta_t|y_{1:t}) \propto \pi(y_t|\theta_t)\pi(\theta_t) = \pi(y_t|\theta_t)\pi(\theta_t|y_{1:t-1}). \tag{1.9}$$

En la igualdad anterior, una buena manera de asignar los priores en el instante t es la predicción de θ en el $t - 1$ basando las informaciones $i = 1 : (t - 1)$.

Distribución del filtrado y la predición La distrbución 1.9 se corresponde al problema de filtrado sin el término constante $1/\pi(y_t|y_{1:t-1})$, la normalización. Para computarla en el instante t, es necesario conocer la predición tanto θ como y en el instante anterior. Otra forma más apropiada de demostrar el resultado anterior es considerar $\pi(\theta_t|y_{1:t}) = \pi(\theta_t|y_{1:t-1}, y_t)$ y aplicando el teorema de Bayes en el término y_t:

$$\pi(\theta_t|y_{1:t-1}, y_t) = \frac{\pi(\theta_t|y_{1:t-1})\pi(y_t|\theta_t, y_{1:t-1})}{\pi(y_t|y_{1:t-1})} = \frac{\pi(\theta_t|y_{1:t-1})\pi(y_t|\theta_t)}{\pi(y_t|y_{1:t-1})}$$

donde la última igualdad está usando la propiedad de independencia condicional de y_t.

Teniendo en cuenta la propiedad del estado θ_t, $\pi(\theta_t|\theta_{t-1}; y_{1:t-1}) = \pi(\theta_t|\theta_{t-1})$, la distribución de la predicción de θ_t es

$$\begin{aligned}\pi(\theta_t|y_{1:t-1}) &= \int \pi(\theta_{t-1}, \theta_t|y_{1:t-1})d\theta_{t-1} \\ &= \int \pi(\theta_t|\theta_{t-1}; y_{1:t-1})\pi(\theta_{t-1}|y_{1:t-1})d\theta_{t-1} \\ &= \int \pi(\theta_t|\theta_{t-1})\pi(\theta_{t-1}|y_{1:t-1})d\theta_{t-1}.\end{aligned}$$

Y la predicción de Y_t viene dada por

$$\begin{aligned}\pi(y_t|y_{1:t-1}) &= \int \pi(y_t, \theta_t|y_{1:t-1})d\theta_t \\ &= \int \pi(y_t|\theta_t; y_{1:t-1})\pi(\theta_t|y_{1:t-1})d\theta_t \\ &= \int \pi(y_t|\theta_t)\pi(\theta_t|y_{1:t-1})d\theta_t,\end{aligned}$$

donde la última igualdad usa de nuevo la independencia condicional de y_t.

A partir de las prediciones de un paso, podemos generalizar a k pasos y las distribuciones de predicción son:

$$\pi(\theta_{t+k}|y_{1:t}) = \int \pi(t_k|\theta_{t+k-1})\pi(\theta_{t+k-1}|y_{1:t})|d\theta_{t+k-1}$$

y

$$\pi(y_{t+k}|y_{1:t}) = \int \pi(y_{t+k}|\theta_{t+k})\pi(\theta_{t+k}|y_{1:t})d\theta_{t+k}$$

Distribución del alisado Con la distribución de filtrado, uno se puede estudiar el comportamiento del estado basando las informaciones hasta el mismo instante. ¿Y qué pasa, si queremos hacer una inspección retrospectiva del sistema basando todas las informaciones disponibles? Es decir, dando las informaciones completas $y_{1:T}$, como se distribuyen los estimadores $\theta_t|y_{1:T}$ con $t < T$. Un aplicación fácil de este tipo de análisis es la determinación de la velocidad de un coche por el radar.

Bajo las hipótesis de DLM y dando θ_{t+1} y $y_{1:T}$, la distribución de transition inversa de θ_t es

$$\pi(\theta_t|\theta_{t+1}; y_{1:T}) = \pi(\theta_t|\theta_{t+1}; y_{1:t}). \tag{1.10}$$

Esta igualdad es cierta porque dando θ_{t+1}, $y_{t+1:T}$ son independientes a θ_t y, por tanto, la ecuación se puede simplificar. Aplicando la fórmula de Bayes en el término θ_{t+1}, obtenemos:

$$\pi(\theta_t|\theta_{t+1}; y_{1:t}) = \frac{\pi(\theta_{t+1}|\theta_t; y_{1:t})\pi(\theta_t|y_{1:t})}{\pi(\theta_{t+1}|y_{1:t})} = \frac{\pi(\theta_{t+1}|\theta_t)\pi(\theta_t|y_{1:t})}{\pi(\theta_{t+1}|y_{1:t})}. \tag{1.11}$$

Finalmente, la distribución de alisado, $\pi(\theta_t|y_{1:T})$, se puede calcular mediante la distribución marginal $\pi(\theta_t, \theta_{t+1}|y_{1:T})$:

$$\pi(\theta_t|y_{1:T}) = \int \pi(\theta_t, \theta_{t+1}|y_{1:T})d\theta_{t+1} = \int \pi(\theta_{t+1|y_{1:T}})\pi(\theta_t|\theta_{t+1}; y_{1:T})d\theta_{t+1}. \tag{1.12}$$

Sustituir la expresión de las ecuación 1.10 y 1.11 a 1.12, tenemos:

$$\begin{aligned}\pi(\theta_t|y_{1:T}) = & \int \pi(\theta_{t+1|y_{1:T}})\frac{\pi(\theta_{t+1}|\theta_t)\pi(\theta_t|y_{1:t})}{\pi(\theta_{t+1}|y_{1:t})}d\theta_{t+1} \\ = & \ \pi(\theta_t|y_{1:t}) \int \pi(\theta_{t+1}|\theta_t)\frac{\pi(\theta_{t+1}|y_{1:T})}{\pi(\theta_{t+1}|y_{1:t})}d\theta_{t+1}\end{aligned}$$

Kalman Filter El método recursivo de las distribuciones anteriores sirve para los modelos de espacio y estado con las mismas hipótesis. En el caso de DLM, tanto el prior de estado como la distribución condicionada de observación siguen la distribución normal y, por la teoría de Probabilidad y Processo Estocástico, la distribución del posterior del estado también está dentro de la familia normal por ser conjugada. Por tanto, es suficiente calcular la esperanza y la matriz de varianza de los estimadores.

Dando $\{y_t\}_{t=1:T}$ y suponiendo una distribución prior $\theta_0 \sim N(m_0, C_0)$, el método de KF se puede describirse en el pseudocódigo siguiente:

Algorithm 1: Pseudo-código del Filtro de Kalman

Input : Conjunt de observaciones $y_{1:T}$
Output: Conjunto de distribuciones: predicción, filtrado y alisado

for $i \leftarrow 1$ **to** T **do**
  computar la predición de estado $\theta_{i|i-1}$
  computar la predicción de observación $y_{i|i-1}$
  computar el error de predición de observación $e_i = y_i - y_{i|i-1}$
  computar el filtrado de estado $\theta_{i|i}$
  asignar el filtrado como prior para $i = i + 1$
end
for $i \leftarrow T - 1$ **to** *1* **do**
  computar la distribución alisado $\theta_{i|T}$
end

Como mencionado anteriormente, para hallar las distribuciones es suficiente determinar los parametros, esperanza y varianza. Dando un modelo DLM definido 1.2 y 1.3 y sean m_{t-1} y C_{t-1} la esperanza y la varianza del filtrado $\theta_{t-1}|y_{1:t-1}$ conocidas, entonces la estimación de los parámetros de un paso es el siguiente:

1. La distribución de la predicción de estado sigue $N(a_t, R_t)$ con

$$\begin{aligned} a_t &= \mathbb{E}[\theta_t|y_{1:t-1}] = \mathbb{E}[G_t\theta_{t-1} + w_t|y_{1:t-1}] = G_t\mathbb{E}[\theta_{t-1}|y_{1:t-1}] + \mathbb{E}[w_t] \\ &= G_t \cdot m_{t-1} \\ R_t &= V[\theta_t|y_{1:t-1}] = V[G_t\theta_{t-1} + w_t|y_{1:t-1}] = G_tV[\theta_{t-1}|y_{1:t-1}]G_t' + V[w_t] \\ &= G_t \cdot C_{t-1} \cdot G_t' + W_t. \end{aligned}$$

2. La distribución de la predicción de observación sigue $N(f_t, Q_t)$ con

$$\begin{aligned} f_t &= \mathbb{E}[y_t|y_{1:t-1}] = \mathbb{E}[F_t\theta_t + v_t|y_{1:t-1}] = F_t\mathbb{E}[\theta_t|y_{1:t-1}] + \mathbb{E}[v_t] \\ &= F_t \cdot a_t \\ Q_t &= V[y_t|y_{1:t-1}] = V[F_t\theta_t + v_t|y_{1:t-1}] = F_tV[\theta_t|y_{1:t-1}]F_t' + V[v_t] \\ &= F_t \cdot R_t \cdot F_t' + V_t. \end{aligned}$$

3. La distribución del filtrado de estado sigue $N(m_t, C_t)$ con

$$\begin{aligned} m_t &= a_t + R_tF_t'Q_t^{-1}(Y_t - F_ta_t) \\ C_t &= R_t - R_tF_t'Q_t^{-1}F_tR_t. \end{aligned}$$

La demostración del punto 3 se basa en un variante del teorema de la normal multivariante: dandas dos variables aleatorias $X = aU + bV$ y $Y = cU + dV$ con U y V normales indepedentes, entonces (X, Y) es conjuntamente gausiano. Es suficiente demostar $\forall s = (s_1, s_2)$, $Z = s_1 \cdot X + s_2 \cdot Y$ tambien es normal. Tenemos

$$Z = s_1 \cdot X + s_2 \cdot Y = (a \cdot s_1 + c \cdot s_2)U + (b \cdot s_1 + d \cdot s_2)V$$

y el resultado es directo porque la suma de normal independiente es normal. Como la consecuencia, $(Y_t, \theta_t|y_{1:t-1})$ es conjunatamente normal

$$(Y_t, \theta_t|y_{1:t-1})' \sim N\left(\begin{pmatrix} f_t \\ a_t \end{pmatrix}, \begin{pmatrix} Q_t & \Sigma \\ \Sigma' & R_t \end{pmatrix}\right)$$

donde

$$\begin{aligned} \Sigma &= Cov((Y_t, \theta_t)'|y_{1:t-1}) \\ &= Cov((F_t\theta_t + v_t, \theta_t)'|y_{1:t-1}) \\ &= F_tV[\theta_t|y_{1:t-1}] + Cov((v_t, \theta_t)'|y_{1:t-1}) \\ &= F_tR_t. \end{aligned}$$

Por la propiedad de la normal multivariante, $\theta_t|y_{1:t-1}; Y_t = y_t$ es normal con parámetros

$$\mathbb{E}[\theta_t|y_{1:t-1}; Y_t = y_t] = a_t + \Sigma'Q_t^{-1}(y_t - f_t) = a_t + R_t'F_t'Q_t^{-1}(y_t - F_t \cdot a_t) \qquad (1.13)$$

y

$$V[\theta_t|y_{1:t-1}; Y_t = y_t] = R_t - \Sigma' Q_t^{-1}\Sigma = R_t - R_t' F_t' Q_t^{-1} F_t R_t \tag{1.14}$$

Una vez construida la distribución de los filtrados $\theta_t \sim N(m_t, C_t)_{t=1:T}$, es el momento de computar la distribución de los alisados usando el algoritmo inverso a partir del instante $t = T$.

Sea $\theta_t|y_{1:T}$ normales con párametros (s_t, S_t), donde $t = 1 : T$. Supongamos que la distribución de alisado para el instante $t + 1$ es conocida, entonces para hallar la expresión los parámetros en el instante anterior es necesario utilizar de nuevo la pripiedad de la normal multivariante, $(\theta_t, \theta_{t+1}|y_{1:T})$. Así pues, la esperanza y la varianza son

$$s_t = \mathbb{E}[\theta_t|y_{1:T}] = \mathbb{E}[\mathbb{E}[\theta_t|\theta_{t+1}; y_{1:T}]|y_{1:T}] \tag{1.15}$$

y

$$S_t = V[\theta_t|y_{1:T}] = V[\mathbb{E}[\theta_t|\theta_{t+1}; y_{1:T}]|y_{1:T}] + \mathbb{E}[V[\theta_t|\theta_{t+1}; y_{1:T}]|y_{1:T}]], \tag{1.16}$$

donde en la igualdad de S_t estamos aplicando la ley de la varianza total (*Law of total variance*). Sin embargo, no olvidamos que dando θ_{t+1}, $Y_{t+1:T}$ son independientes a θ_t y los problemas anteriores se reducen a hallar la esperanza y la varianza condicional.

Para aplicar las fórmulas 1.13 y 1.14 es necesario hallar primero

$$\begin{aligned} Cov(\theta_t, \theta_{t+1}|y_{1:t}) &= Cov(\theta_t, G_{t+1}\theta_t + W_{t+1}|y_{1:t}) \\ &= Cov(\theta_t, G_{t+1}\theta_t|y_{1:t}) + Cov(\theta_t, W_{t+1}|y_{1:t}) \\ &= V[\theta_t|y_{1:t}]G_{t+1}' \\ &= C_t G_{t+1}'. \end{aligned}$$

Por consiguiente,

$$\begin{aligned} \mathbb{E}[\theta_t|\theta_{t+1}; y_{1:t}] &= \mathbb{E}[\theta_t|y_{1:t}] + Cov(\theta_t, \theta_{t+1}|y_{1:t})V[\theta_{t+1}|y_{1:t}]^{-1}(\theta_{t+1} - \mathbb{E}[\theta_{t+1|y_{1:t}}]) \\ &= m_t + C_t G_{t+1}' R_{t+1}^{-1}(\theta_{t+1} - a_{t+1}) \\ V[\theta_t|\theta_{t+1}; y_{1:t}] &= V[\theta_t|y_{1:t}] - Cov(\theta_t, \theta_{t+1}|y_{1:t})V[\theta_{t+1}|y_{1:t}]^{-1}Cov(\theta_t, \theta_{t+1}|y_{1:t})' \\ &= C_t - C_t G_{t+1}' R_{t+1}^{-1} G_{t+1} C_t, \end{aligned}$$

donde m_t y C_t son la esperanza y la varianza del filtrado en t, a_{t+1} y R_{t+1} de la predicción en $t + 1$ y G_{t+1} la matriz de evolución. Sustituimos estas dos expresiones

a las ecuaciones 1.15 y 1.16, tenemos

$$\begin{aligned}
s_t &= \mathbb{E}[m_t + C_t G'_{t+1} R^{-1}_{t+1}(\theta_{t+1} - a_{t+1})|y_{1:T}] \\
&= m_t + C_t G'_{t+1} R^{-1}_{t+1}(\mathbb{E}[\theta_{t+1}|y_{1:T}] - a_{t+1}) \\
&= m_t + C_t G'_{t+1} R^{-1}_{t+1}(s_{t+1} - a_{t+1}) \\
S_t &= V[m_t + C_t G'_{t+1} R^{-1}_{t+1}(\theta_{t+1} - a_{t+1})|y_{1:T}] + \mathbb{E}[C_t - C_t G'_{t+1} R^{-1}_{t+1} G_{t+1} C_t] \\
&= C_t G'_{t+1} R^{-1}_{t+1} V[\theta_{t+1}|y_{1:T}] R^{-1}_{t+1} G_{t+1} C_t + C_t - C_t G'_{t+1} R^{-1}_{t+1} G_{t+1} C_t \\
&= C_t G'_{t+1} R^{-1}_{t+1} S_{t+1} R^{-1}_{t+1} G_{t+1} C_t + C_t - C_t G'_{t+1} R^{-1}_{t+1} R_{t+1} R^{-1}_{t+1} G_{t+1} C_t \\
&= C_t - C_t G'_{t+1} R^{-1}_{t+1}(S_{t+1} - R_{t+1}) R^{-1}_{t+1} G_{t+1} C_t
\end{aligned}$$

En resumen, la estimación de los parámetros del método del Filtro de Kalman se detallen en la figura siguiente

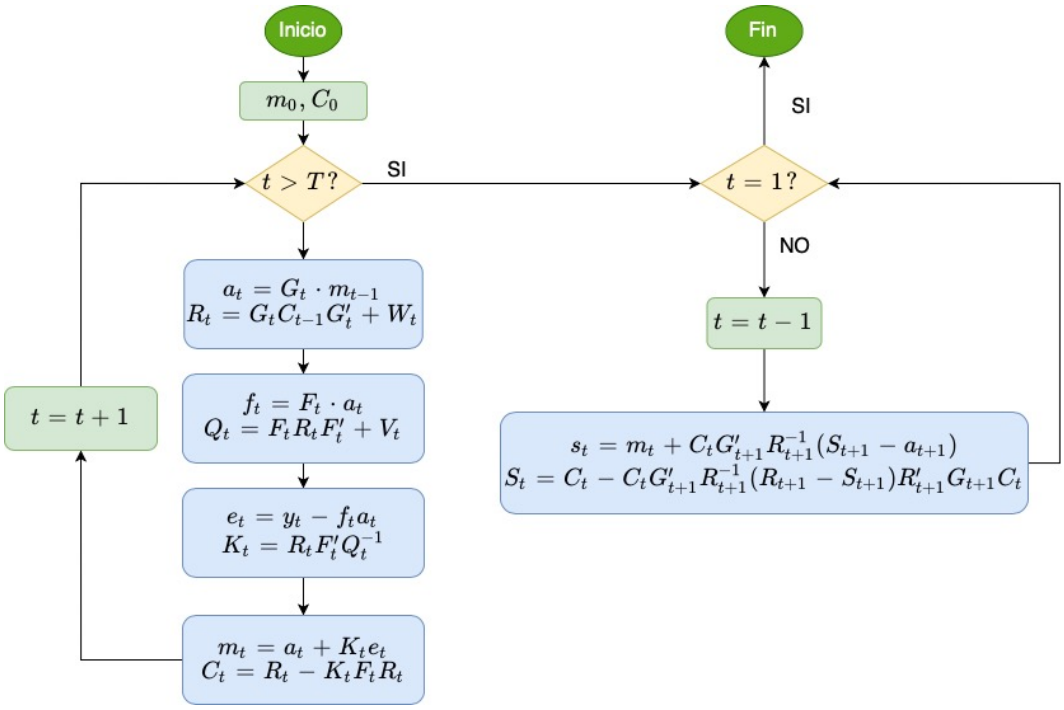


Figura 1.3: Flujo del algoritmo KF

1.1.3. Proceso de inovación y validación del modelo

Por la definición, una serie temporal es una secuencia de datos ordenados cronológicamente y, en general, presenta una correlación entre ellos. Es decir, no existen muestras simples en este sentido. Por la presencia de autocorrelación, no es posible de encontrar una factoriazación total de la distribución conjunta de la muestra, pero sí acepta una factorización condicional en muchos casos:

$$f_{Y_1,\cdots,Y_T}(y_1,\cdots,y_T) = f_{Y_1}(y_1)\cdot f_{Y_2|Y_1}(y_2)\cdot f_{Y3|Y_1,Y_2}(y_3)\cdots \neq \prod_{i=1}^{T} f_{Y_i}(y_i).$$

Sin embargo, esta factorización complicada nos lleva a una fórmula no análitica y, normalmente, no cerrada. Para poder ser analizada tendremos que encontrar una forma tal que la conjunta se pueda admitir una descomposición total.

Una posible solución es usar los errores de la predcción, $\{e_t\}$, también conocidos como el proceso de innovación. Estos errores se puede definirse tanto en término de observación como en estado:

$$\begin{aligned} e_t &= y_t - \mathbb{E}[y_t|y_{1:t-1}] &&= y_t - f_t \\ &= y_t - F_t a_t &&= y_t\theta_t + v_t - F_t a_t \\ &= F_t(\theta_t - a_t) + v_t. \end{aligned}$$

Por su definición, e_t

1. Tiene la esperanza nula,

$$\mathbb{E}[e_t] = \mathbb{E}[y_t - f_t] = \mathbb{E}[\mathbb{E}[y_t - f_t|y_{1:t-1}]] = \mathbb{E}[y_t|y_{1:t-1}] - f_t = f_t - f_t = 0.$$

2. Es independiente respecto a $y_s \quad \forall s < t$, de hecho, lo es respecto a cualquiera función de $y_1,\cdots,y_{t-1}$. Sea $Z = f(y_1,\cdots,y_{t-1})$, entonces

$$\begin{aligned} Cov(e_t, Z) &= \mathbb{E}[e_t Z] = \mathbb{E}[\mathbb{E}[e_t Z|y_{1:t-1}]] \\ &= \mathbb{E}[\mathbb{E}[e_t|y_{1:t-1}]Z] = 0. \end{aligned}$$

3. Es una función lineal de $y_{1:t}$. $e_t = y_t - f_t$ y $f_t = \mathbb{E}[y_t|y_{1:t-1}]$ es una función lineal de $y_1,\cdots,y_{t-1}$ por que $\{y_t\}$ son normales y, por tanto, e_t es una función lineal de $y_{1:t}$.
4. Como consecuencia de 2 y 3, $\forall s < t$, $Cov(e_s, e_t) = 0$.
5. Finalmente, el proceos de innovación $\{e_t\}$ es un proceso de Gauss con varianza

$$V[e_t] = V[y_t - y_t|y_{1:t-1}] = V[y_t|y_{1:t-1}] = Q_t.$$

   Si los errores son univariante, en el nuestro caso lo es, $\{\tilde{e}_t\} = \{e_t/Q_t\}$ es una secuencia de normal estándar.

La estimación de parámetros en el término de innovación se describe en la figura 1.3. Un otro término descrito en la misma figura es la matriz de la ganancia de Kalman, K_t. Esta matriz también presenta algunas propiedades intersantes que nos indica cómo tendremos que corregir el estado basando las medidas hechas. Sin embargo, no iremos a explorarlas en este trabajo.

Basando las propiedades del proceso de innovación, para verificar el correcto uso del filtro a un conjunto de datos de series temporales es suficiente analizar los errores originales o normalisados de la predicción. A la continuación, presentaremos tres tipos de tests basando en estas propiedades.

Intervalo de confianza con el nivel 95 % Sabemos que si el filtro es correcto, entonces e_t es una normal con media 0 y error estándar $\sqrt{Q_t}$. Por tanto, para verificar la consistencia del modelo, podemos computar la banda de confianza del 95 %. Es decir, si tenemos estimaciones puntuales de las innovaciones, habrán una probabilidad de 0.95 caen en la siguiente banda $(-1{,}96 \times \sqrt{Q_t}, 1{,}96 \times \sqrt{Q_t})$. En la prática, tenemos que calcular las matrices de varianza de la predicción de observación Q_t y los errores de preddición $y_t - y_t|y_{1:t-1}$. Como que estos errores son estimaciones puntuales de las innovaciones teóricas, entonces 95 % de las estimaciones deberian estar compredida por las cotas definidas anteriormente. La figura siguiente ilustra un ejemplo de aplicación donde el 91 % de los errores está la banda definida.

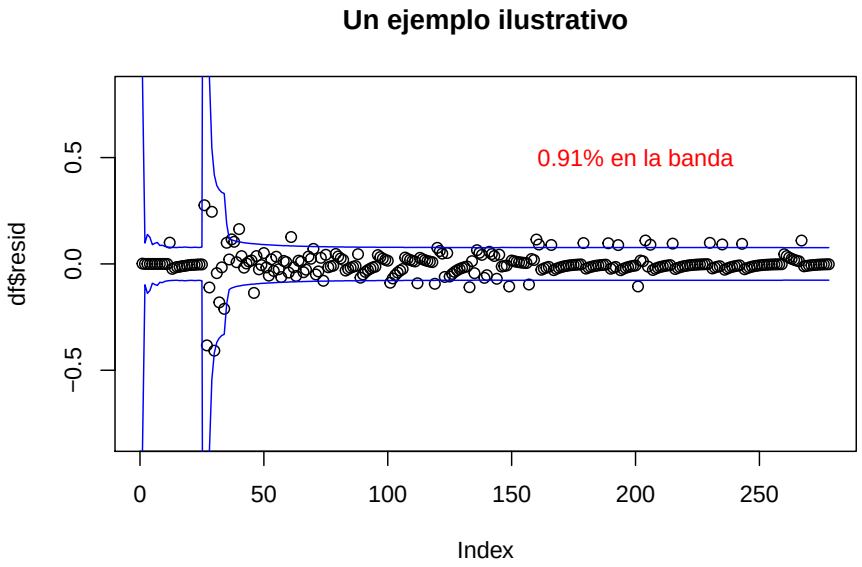


Figura 1.4: Banda de confianza de 0.95

Test de χ^2 para los errores estandarizados Si el filtro es correctamente aplicado, entonces la secuencia de transformaciones $\tilde{e}_t = e_t/Q_t$ es de la normal con media 0 y desviación típica 1 y la suma sigue una distribución de Peason de grado T

$$\sum_{t=1}^{T} \tilde{e}_t^2 \sim \chi_T^2.$$

Por tanto, es suficiente calcular el intervalo centrado de 95 % de la distribución χ_T^2. Además, por la ergodicidad[6] de e_t, la media móvil de las inovaciones

$$\bar{e} = \sum_{t=1}^{N} \tilde{e}_t^2 \to 1,$$

si N suficientemente grande.

[6]Decimos que un proceso estocástico es ergódico si el comportamiento estadístico medio es el mismo a través del tiempo.

Para el ejemplo anterior, el intervalo correspondiente es (233.7, 326.1) y la suma de errores cuadrados es 277. En este caso, no rechazamos la hipótesis nula.. De la misma idea, podemos pintar los $\tilde{e}_t^2$ y la media móvil a través de tiempo en la figura 1.5. Obviamente, si el tamaño crece, la media móvil se tiende a uno y, por tanto, las hipótesis se aceptan en este caso.

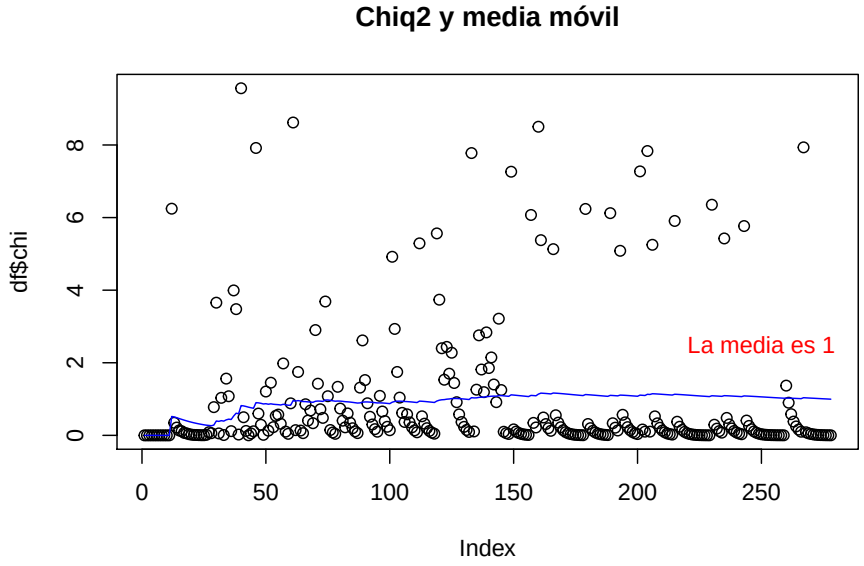


Figura 1.5: Test de Chiq2 de grado 1

Test de la autocorrelación Los primeros tests están asociados a la normalidad de los errores y el último tiene que ver con la independencia. Aquí tenemos dos alternativos, estadístico de Ljung-Box (LB test) y Durbin-Waston (DW test). Sin embargo, para las series temporales, la prueba de Ljung-Box no es una eina correcta de probar la independencia de los errores. El DW test es un estadístico usado para detectar la presencia de autocorrelación de lag 1. Si e_t es el error de la forma $e_t = \rho e_{t-1} + u_t$, la hipótesisi nula para el estadístico DW es $\rho = 0$ y el test de estadístico es

$$d = \frac{\sum_{t=2}^{T}(e_t - e_{t-1})^2}{\sum_{t=1}^{T} e_t^2}.$$

Este estadístico es aproximadamente igual a $2(1-\hat{\rho})$ donde el $\hat{\rho}$ es la autocorrelación muestral de los residuos. En el caso de independencia, el $\rho = 0$ indica un valo de $d = 2$. Si $d < 2$ indica una correlación positiva y el caso contrario una correlación negativa. Sin embargo, para ver la significación del estadístico DW hay que compara con los cotas $d_{L,\alpha}$ y $d_{U,a}$ que varian respecto al tamaño de muesta. Afortunadamente, este estadístico está implementado en *R*: `dwtest` del paquete `lmtest` o `durbinWatsonTest` de `car`.

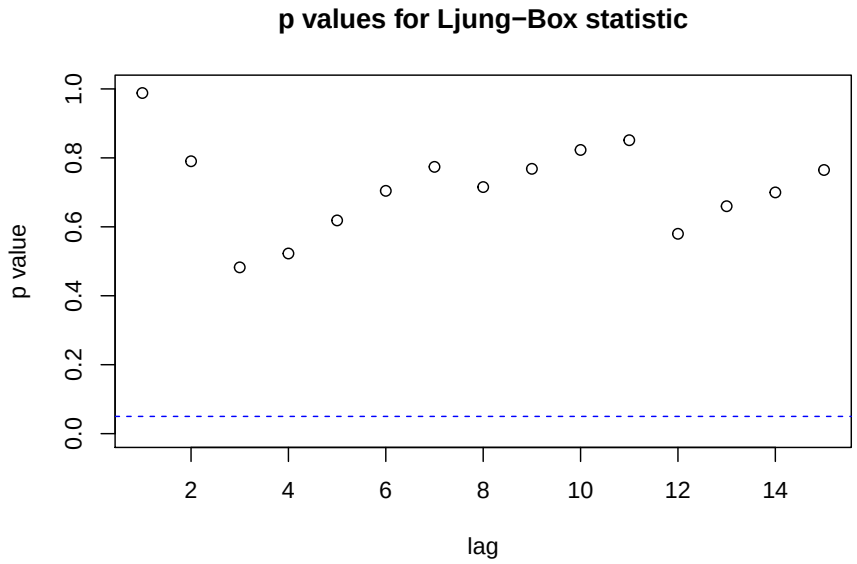
Podemos aplicar tanto LB test como DW test para el ejemplo anteror:

1. test de Durbin-Watson:

```
Durbin-Watson test
```

```
data:  y ~ x
DW = 1.9946, p-value = 0.4861
alternative hypothesis: true autocorrelation is greater than 0
```

2. test de Ljung-Box



1.1.4. Aplicación de DLM: Análisis de la salinidad de la tierra

La sal o el cloruro de sodio es un elemento indispensable tanto para los seres humanos como los cultivos. No obstante, una acumulación de sal en alrededor de la zona de raíz de un cultivo puede dañar las plantas, reducir la producción e incluso cambiar la estructura del suelo causando un daño a largo plazo. Con el fin de preservar la productividad, el control de sal ha sido una tarea necesaria. Un estudio de la conductividad eléctrica de agua del suelo es una buena manera de medir la salinidad, porque la agua pura no conduce la electricidad. Actuamente, existen sensores para la medición de la conductividad de agua usando el modelo de Hilhorst determinístico.

El modelo de Hilhorst [2000] es un modelo donde la correlación entre la conductividad eléctrica (EC: *electrical conductivity*) y el producto de la permitividad dieléctrica relativa (RDP: *relative dielectric permittivity*) de agua y la EC de tierra es lineal. Según a Hilhorst, la EC de agua se puede determinarse a partir de la ecuación:

$$\sigma_p = \frac{\epsilon_p \sigma_b}{\epsilon_b - \epsilon_{\sigma_b=0}}, \tag{1.17}$$

donde

1. σ: la conductividad eléctrica;
2. ϵ: la permitividad dieléctica relativa;
3. p: contenido de agua en la tierra;

4. b: sólido o suelo.

Los valores ϵ_b (no tiene unidad) y σ_b (dS/m) se puede obtener directamente en el suelo a partir de un sensor dieléctrico; ϵ_p es una función de la temperatura[7]; y $\epsilon_{\sigma_b=0}$ se aparece como un offset de la relación linear que depende del tipo de suelo y hace referencia a la permitividad cuando la conductividad eléctrica es cero. Para tener un modelo determínistico de Hilhorst es suficiente fijar el valor de offset. Según el estudio de Hilhorst, $\epsilon_{\sigma_b=0}$ varia entre 1.9 y 7.6 y un valor general igual a 4.1 fue recomendado por él. Muchos estudios usando este modelo determinístico en sus experimentos con un offset distinto. Por ejemplo,el manual de 5TE recomenda un valor 6.

En el mundo real, es díficil de considerar todas las covariantes de un determinado fenómeno y, por tanto, el uso de un modelo determinístico tiene su defecto. Dando al modelo un componente aleatoria, Basem et al. [2018] introdujo el uso del DLM al análisis de σ_p usando una serie de transformaciones a partir del modelo de Hilhorst:

1. Despejar ϵ_b de la ecuación 1.17,

$$\epsilon_b = \epsilon_{\sigma_b=0} + \frac{\epsilon_p \sigma_b}{\sigma_p}. \tag{1.18}$$

2. Considerar los cambios $y_t = (\epsilon_b)_t$, $\theta_t = (\theta_1, \theta_2)_t = (\epsilon_{\sigma_b=0}, \sigma_p^{-1})'_t$ y $F_t = (1 \quad \epsilon_p \sigma_b)_t$, la ecuación 1.18 se puede reescribirse como

$$y_t = F_t \cdot \theta_t.$$

3. Añadir la ecuación de la evolución de estado θ_t como un paseo aleatorio,

$$\begin{pmatrix}\theta_1 \\ \theta_2\end{pmatrix}_t = \begin{pmatrix}\theta_1 \\ \theta_2\end{pmatrix}_{t-1} + \begin{pmatrix}w^{(1)} \\ w^{(2)}\end{pmatrix}_t, \quad \begin{pmatrix}w^{(1)} \\ w^{(2)}\end{pmatrix}_t \sim N\left(\begin{pmatrix}0 \\ 0\end{pmatrix}_t, \begin{pmatrix}q_{11} & 0 \\ 0 & q_{22}\end{pmatrix}\right).$$

4. Añadir un ruido blanco v_t debido al error de medición a la ecuación de observación

$$y_t = F_t \cdot \theta_t + v_t, \quad v_t \sim N(0, V).$$

5. Finalmente, obtenemos un DLM con

$$\theta_t = \begin{pmatrix}\sigma_b = 0 \\ \sigma_p^{-1}\end{pmatrix}_t, \quad W = \begin{pmatrix}q_{11} & 0 \\ 0 & q_{22}\end{pmatrix}, \quad F_t = (1 \quad \epsilon_p \sigma_b)_t \quad \text{y} \quad G_t = G = \begin{pmatrix}1 & 0 \\ 0 & 1\end{pmatrix}.$$

Este modelo considera el offset y la EC de agua como variables latentes no medibles directamente, pero se puede obtenerlas a partir de las observaciones. Aplicando el filtro de Kalman, poderemos hallar una buena estimación usando el estimador de alisado, el estimador que tiene menos variación. Actualmente, no existen estudios de offset como una serie temporal, más habitual se lo trata como un valor constante ($q_{11} = 0$).

[7] $\epsilon_p = 80{,}3 - 0{,}37(T_b - 20)$, donde T_b indica la temperatura de la tierra en centigrado.

Este trabajo es una extensión del Basem et al. [2018], la construcción de un paquete en R basando en su análisis. Un paquete es una integración de varias funciones u objectos con el fin de dar un proceso estándar y facilitar a los usuarios. En el próximo capítulo exploraremos el funcionamiento de un paquete en R y, más concreto, la libreria basada en los objetos de `S4`.

1.2. Red Neuronal Recurrente

En esta sección estudiaremos algunos puntos de la Red Neuronal Recurrent (RNN: *Recurrent Neural Network*):

1. las redes neuronales artificiales, el uso de las distintas funciones de activación;
2. la necesidad de tener una extensión como RNN, su configuración, el número total de parámetros del modelo definido en el package `sm4sd`,
3. y, sobre todo, la aplicación del Filtro Kalman Extendido (EKF: *Extented kalman Filter*) como el optimizador del RNN

1.2.1. Redes Neuronales Artificiales

Las redes neuronales artificiales son los modelos de apredndizaje automático inspirados por el celebro humano. En general, una red neuronal artificial es un terno complejo

$$(\{l_i^{(k)}, l_i^{(k)}(\cdot)\}_{i\in I_k}, \{w_{ij}^{(k)}\}_{i\in I_{k-1}, j\in I_k})$$

donde

1. $l_i^{(k)}$ representa el nudo i de la capa k;
2. $l_i^{(k)}(\cdot)$ es la función de activación o de link asociada al nudo $l_i^{(k)}$ y para la $k = 1$, la función de link es la identidad;
3. $w_{ij}^{(k)}$ indica el peso asociado a la conexión desde el nudo $l_i^{(k-1)}$ al $l_j^{(k)}$[8];
4. k es la profundidad o la capa de una red y I_k, el conjunto de los posibles nudos en la profundidad k.

Los nudos se refieren a las neuronas biológicas que se encargan de recibir y enviar las informaciones y el motor de cada uno es la función de link, que transforma las informaciones. Las posibles conexiones entre estos nudos de dos capas consecutivas se denotan por las flechas, que representan intuitivamente las sinapsis. Para el caso de las redes normales, no deben aparecer ciclos cerrados y las flechas siempre apuntan hacia a la capa posterior. Dependiendo la posición de estas capas podemos obtener

[8] Aquí, la index i se usa para el nudo de la capa superior y j, de la capa posterior.

3 tipos: la capa de entrada (*input layer*), la capa oculta (*hidden layer*) y la capa de salida (*output layer*)[9]. Finalmente, la complejidad de una red se define a través del número total de los nudos y de las conexiones, la cardinal del conjunto $w_{ij}^{(k)}$. Una red simple con dos capas ocultas, tres nodos de entradas y un nodo de salida se muestra en la figura siguiente:

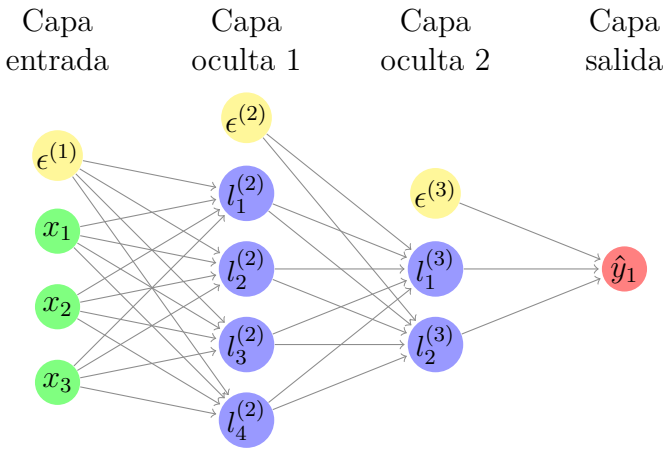


Figura 1.6: Una red simple con dos capas ocultas.

El valor transmitido de la nodo j de la capa k, se denota por $v_j^{(k)}$, se calcula aplicando la función de link a la suma ponderada de los nodos de la capa anterior junto con el término de *intercept*[10]:

$$v_j^k = l_j^{(k)} \left(\sum_{i \in I_{k-1}} w_{ij} \cdot v_i^{(k-1)} + \epsilon^{(k-1)} \right) \quad \forall j \in I_k. \tag{1.19}$$

Por conveiniencia, podemos dividirla en dos pasos asignando la activación entrada $a_j^{(k)}$ como la suma de ecuación 1.19:

$$v_j^{(k)} = l_j^{(k)} \left(a_j^{(k)} \right) \quad \text{y} \quad a_j^{(k)} = \sum_{i \in I_{k-1}} w_{ij} \cdot v_i^{(k-1)} + \epsilon^{(k-1)} \quad \forall j \in I_k. \tag{1.20}$$

La figura siguiente representa la configuración de una determinada neurona j de la capa k:

[9] La primera capa es de entrada y la última es de salida, las restas capas entremedias son capas ocultas.

[10] El término $\epsilon^{(k-1)}$ de la figura 1.6 es el *intercept*, juega el mismo papel en el caso de regresión, y su peso asociado siempre es la unidad.

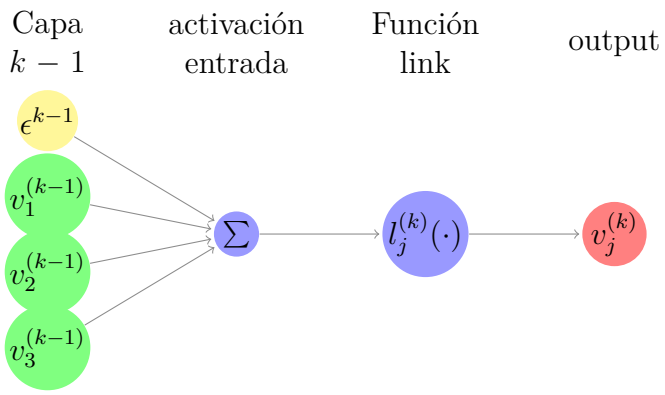


Figura 1.7: Mecanismo de una neurona artificial.

La mayoria de las funciones de activación es no lineal y el uso de estas funciones hace que la red sea capaz de capturar los patrones a partir de los datos y, además, realizar tareas más complejas. No obstante, como otros modelos de aprendizaje automático, no existe ninguna teoría que explica explícitamente el funcionamiento de las capas ocultas. Por este motivo, estos modelos también se conocen como modelos de caja negra. Las activaciones más comunes en las redes son:

1. lineal, $l(z) = a \cdot z$;
2. sigmoide o logística, $\sigma(z) = (1 + e^{-z})^{-1}$;
3. tangente hiperbólica, $\tanh(z) = (e^z - e^{-z})/(e^z + e^{-z})$;
4. ReLu (*Rectified Linear Unit*), $l(z) = max(0, z)$;
5. y *SoftMax*.

Dos problemas principales de usar las activaciones lineales en las capas son: 1. la imposibilidad de usar el algoritmo del gradiente descendente[11], 2. el colapso de todas las capas en una única capa y, debido a esto, la respuesta será la combinación lineal de las entradas. Por estas inconveniencias, las activaciones lineales suelen usar en la capa de salida.

Las activaciónes σ y tanh comparten algunas similaridades, forma de S con gradiente suave. La image de la primera es $(0, 1)$ centrado a $1/2$ y los valores $|z| > 2$ llevan una predicción cerca a 0 o 1. Por estas propiedades, esta activación es usualmente usado en la capa de salida para los problemas de clasificación binaria. Un valor de predicción superior a 0,5 indica el dato de entrada corresponde a la clase objetivo. Al contrario, el rango de tanh es -1 a 1 centrado a 0 y normalmente se usa en las capas ocultas. Estas dos activaciones también provocan algunas situaciones no deseables Por ejemplo, para los valores extremos, no habrá cambio siginificativo de la predicción y, por tanto, causan el problema de desvanecimiento de gradiente (*Vanishing Gradient problem*).

Comparando con las dos activaciones anteriores, la función *ReLu* tiene un coste computacional más ligero por su simplicidad. Además, admite el algoritmo de *back-*

[11]El gradiente descendente es un método iteractivo de optimización de primer order para hallar el mínimo de una determinada función. El problema de este algoritmo es la convergencia local.

propagation para calcular los errores. Y la función de *Softmax* es como una extensión de σ para los problemas de clasificación multiple.

1.2.2. La necesidad de RNN y su configuración

Las redes artificiales tienen un uso muy amplio: el reconocimiento de escritura a mano, el problema de TSP (*Traveling Salesman problem*), la compresión de imagenes y etc. Sin embargo, a pesar de su poder, las redes estándares tiene su limitación. Como a los modelos de regresión, la red estándar se basa en la indenpendencia de las muestras y después de procesar todos los datos, el estado latente se pierde. Sin embargo, para los datos secuenciales o tiene un componente temporal o espacial, esto no es aceptable. La necesidad de capturar las informaciones secuendarias a través de los datos para tener una predcción precisa hace que las redes se evolucionan.

El modelo de las redes recurrentes, **RNN**, es una solución incluyendo unas capas recurrentes denotadas por $\boldsymbol{h}$. Los nudos de estas capas interconectan y autoconectan a través del tiempo o instancias para transmitir las informaciones latentes. De alguna menera, introduce la noción de tiempo al modelo y el calculo de los valores de v_j de la capa recurrente incluye su valor en el instante anterior.

Actualemente, las arquitecturas de RNN más usadas son *LSTM* (*Long short-term memory*) y RNN bidireccional (BRNNs). Sin embargo, el estudio de las redes no es el objetivo principal de esta tesis y, para añadir la diversidad del modelo, he considerado el modelo de RNN simple para la aprendizaje automática. Dicemos que una RNN es simple si solo contiene una única capa oculta y una única capa recurrent con el número de nodos igual al número de entradas. Además, la conexión entre la capa de entrada y la capa oculta es simple. Para la figura 1.8, si $v_j^{(l)}$ representa la salida del nudo j de la capa oculta, entonces

$$v_j^l = l_j(w_j \cdot x_j + e_j^l),$$

donde el superíndece l indica la capa oculta. Para el nudo j de la capa recurrente, el cálculo el más complejo:

$$v_{j,t}^{(h)} = h_j \left(\sum_{i=1}^{3} \left(W_{i,j}^{lh} \cdot v_{i,t}^{(l)} + W_{i,j}^{hh} v_{i,t-1}^{(h)} \cdot \right) + \epsilon_j^{(h)} \right),$$

donde $W_{i,j}^{lh}$ es el peso asociado a la conexión entre l_i y h_j y $W_{i,j}^{hh}$ indica el peso entre h_i en el instante anterior a h_j. Añadiendo el componente temporal, nos obliga a usar la segunda subíndex para indicar el tiempo.

Para este tipo de modelo (ver figura 1.8), nos interesa calcular el número de parámetros definidos para tener una idea de su complejidad. sea s el número de nudos de la capa entrada, el número de párametros en las distintas capas son:

- Capa oculta: un peso y un error para cada nudo de esta capa $(s + s)$.
- Capa recurrente: como que la conexión es completa, tenemos s^2 conexiones entre la capa oculta y la capa recurrente y otro s^2 entre la capa recurrente en instante $t - 1$ a t. Finalmente, tenemos s errores $(2s^2 + s)$.
- Capa salida: s conexiones y un error $(s + 1)$.

En total, tenemos

$$C = s + s + s^2 + s^2 + s + s + 1 = 2s^2 + 4s + 1.$$

Si $s = 3$, tenemos que estimar un total de 31 parámetros[12].

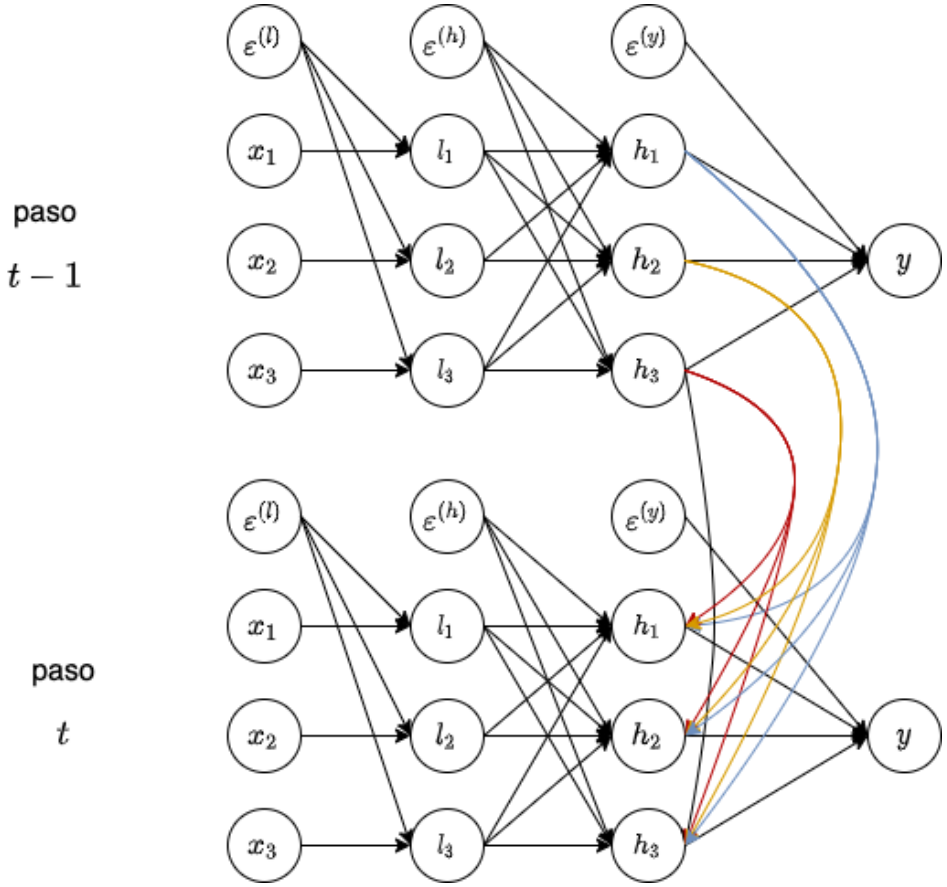


Figura 1.8: Ejemplo de una red recurrent en t-1 a t

1.2.3. Extensión de Kalman Filter

El filtro de Kalman extendido (EKF: *Extented Kalman Filter*) es una versión no lineal del KF que transforma las ecuaciones originales no lineales por la expansión de Taylor de primer order. Debido a esta aproximación lineal, este filtro es una solución sub-óptima para este tipo de problema Es decir, el EKF puede converger en una óptima local o diverger en algunos casos como el método de **Newton-Raphson**. Consideramos la extesión de la ecuación 1.1,

$$\begin{aligned} y_t &= h_t(\theta_t) + v_t \qquad v_t \sim N(0, \Sigma_{v_t}), \\ \theta_t &= g_t(\theta_{t-1}) + w_t \quad w_t \sim N(0, \Sigma_{w_t}). \end{aligned} \tag{1.21}$$

[12] A veces, estos modelos son demasiados simples hasta que no se pueden caputar suficientemente las informaciones lantentes por el escaso de los parámetros. Una buena manera es transformar los inputs añadiendo la ventaja temporal. Con una venta de 3, el número de parámetro se aumenta a 337.

Supongamos que estas dos secuencias $\{h_t\}$ y $\{g_t\}$ son diferenciales, al menos de orden uno. Dando un t, la expansión de Taylor de las funciones h_t y g_t en los puntos $a_t = \mathbb{E}[\theta_t|y_{1:n-1}]$ y $m_{t-1} = \mathbb{E}[\theta_{t-1}|y_{1:n-1}]$, respectivamente, son

$$\begin{aligned} h_t(\theta_t) &= h_t(a_t) + H_t \cdot (\theta_t - a_t) + \cdots \\ g_t(\theta_{t-1}) &= g_t(m_{t-1}) + G_t \cdot (\theta_{t-1} - m_{t-1}) + \cdots , \end{aligned}$$

donde H_t y G_t son matrices jacobianas definidas como:

$$H_t = \left.\frac{\partial h_t}{\partial \theta_t}\right|_{\theta_t = a_t} \quad \text{y} \quad G_t = \left.\frac{\partial g_t}{\partial \theta_{t-1}}\right|_{\theta_{t-1} = m_{t-1}} .$$

Ignorando los términos superiores, la ecuación 1.21 se puede aproximar por

$$\begin{aligned} y_t &= H_t \cdot \theta_t + h_t(a_t) - H_t \cdot a_t + u_t \\ \theta_t &= G_t \cdot \theta_{t-1} + g_t(m_{t-1}) - G_t \cdot m_{t-1} + v_t. \end{aligned}$$

Usando las mismas terminologías de KF, el método de EKF de un paso se describe en el siguiente: dando m_{t-1} y C_{t-1} la esperanza y varianza del filtrado de θ en $t-1$, entonces

1. Predicción de θ_t:
$$\begin{aligned} a_t &= g_t(m_{t-1}) \\ R_t &= G_t \cdot C_{t-1} \cdot G_t' + \Sigma_{v_t}. \end{aligned}$$
2. Predicción de y_t:
$$\begin{aligned} f_t &= h_t(a_t) \\ Q_t &= H_t \cdot R_t \cdot H_t' + \Sigma_{u_t}. \end{aligned}$$
3. La innvoación y la ganancia de Kalman:
$$\begin{aligned} e_t &= y_t - h_t(a_t) \\ K_t &= R_t \cdot H_t' \cdot Q_t^{-1}. \end{aligned}$$
4. El filtrado de θ_t:
$$\begin{aligned} m_t &= a_t + K_t \cdot e_t \\ C_t &= R_t - K_t \cdot H_t \cdot R_t. \end{aligned}$$

El uso del EKF como el algoritmo de entrenamiento para a las redes artificiales no recurrentes, en las tareas supervisadas, fue originado por Singhal & Wu [1989] y el resultado fue favorable. Poco después, la aplicación a RNN fue hecha por Williams y Zipser en el año 1990. Mientras que las formulaciones clásicas de entrenamiento de las redes recurrents ajustan los pesos respecto a la pérdidad usada, el EKF intenta

a estimar tanto el peso de la red $\hat{W}$ como las actividades de todos los nudos. Por la incoviniencia, reformulamos una red recurrent para facilitar la aplicación.

Sea una red recurrente formada por n_U nudos y n_W pesos. El conjunto de los nudos denota por el vector $U = (U_o, U_h)$ contiene n_o nudos de salidas (U_o) y $n_U - n_o$ de las capas ocultas (U_h). Sea $\theta_t \in \mathbb{R}^{n_U}$ el vector de salida de U en el instante t y x_t el vector de entrada. Sea W_t un vector de los pesos en el instante t que es una vectorización de la matriz (w_{ij}), donde w_{ij} indica el peso del nudo U_i al nudo U_j.

Una vez reformulado, el vector de estado que queremos estimar es $(\theta, W)^t$ y su dinámico sin el componente aleatorio viene dado por

$$\begin{array}{rcl} \theta_k & = & f(\theta_{k-1}, W_{k-1}, x_k) \\ W_k & = & W_{k-1} \end{array}, \tag{1.22}$$

donde f representa el mapeo de la transición general. La linealización de este dinámico es

$$F_k = \begin{pmatrix} F_{1,k} & F_{2,k} \\ 0 & I \end{pmatrix} \quad \text{con} \quad F_{1,k} = \frac{\partial \theta_k}{\partial \theta_{k-1}} \text{ y } F_{2,k} = \frac{\partial \theta_k}{\partial W_{k-1}}.$$

dond $F_{1,k}$ y $F_{2,k}$ son bloques de dimensión $n_U \times n_U$ y $n_U \times n_W$ respetivamente. La ecuación de observación es la proyección ortogonal de los primeros n_o componente, denota por H, del estado:

$$y_t = H \cdot \begin{pmatrix} \theta_t \\ W_t \end{pmatrix}.$$

De la manera similar, podemos descomponer la matriz de covarianza del estado y la matriz de la ganancia de Kalman en los bloques

$$P = \begin{pmatrix} P_1 & P_2 \\ P3 = P_2' & P_4 \end{pmatrix} \quad K = \begin{pmatrix} K_1 \\ K_2 \end{pmatrix},$$

donde K_1 describe el cambio respecto a las actividades de la red y K_2 los pesos.

Dandos los valores iniciales $\theta_0 = (1/2, \cdots, 1/2)^t$, $P_{1,0} = \sigma_u^2(0) \cdot I$, $P_{4,0} = \sigma_W^2(0) \cdot I$, $P_2 = 0$ y W_0 un vector aleatorio con valores pequeños. Por la simetría de la matrix P, no es necesario de actualizar P_3. El método EKF a RNN descrito por Williams y Zipser es el siguiente

Algorithm 2: Algoritmo de EKF a RNN

for $t \leftarrow 1$ **to** T **do**

Estimación de estado

v_t = concadena $\hat{\theta}_{t-1|t-1}$ y x_t # x_t el input en t

$\hat{\theta}_{t|t-1} = g\left(\hat{W}_{t-1|t-1} \cdot v_t\right)$

$\hat{W}_{t|t-1} = \hat{W}_{t-1|t-1}$

Linealizar el dinámico

para cada peso w_{ij}: $F_{1,t}(i,j) = \hat{y}_{i,t|t-1} \cdot (1 - \hat{y}_{i,t|t-1}) \cdot \hat{W}_{(i,j),t|t-1}$

para cada nudo y_i y peso w_{jl}: $F_{2,t}(i, \{j,l\}) = \delta_{ij} \cdot \hat{\theta}_{i,t|t-1} \cdot (1 - \hat{\theta}_{i,t|t-1}) \cdot v_l$

Estimación de covarianza

$P_{4,t|t-1} = P_{4,t-1|t-1} + \sigma_W^2 \cdot I$

$P_{2,t|t-1} = F_1 \cdot P_{2,t-1|t-1} + F_2 \cdot P_{4,t-1|t-1}$

$P_{1,t|t-1} = (F_1 \cdot P'_{2,t-1|t-1}) \cdot F'_1 + P_{2,t-1|t-1} \cdot F'_2 + \sigma_U^2 \cdot I$

Ganancia de Kalman

L_1 = primeros n_o columnas de $P_{1,t|t-1}$

L_2 = primeros n_0 columnas de $P'_{2,t|t-1}$

M = los primeros n_o columnas y filas de $P_{1,t-1|t-1}$

$N = (M + H(\sigma_U^2) \cdot I)^{-1}$

$K_1 = L_1 \cdot N$

$K_2 = L_2 \cdot N$

filtrado de estado

$e_t = z_t - \hat{\theta}_{t|t-1}$ # z_t la observación en t

$\hat{\theta}_{t|t} = \hat{\theta}_{t|t-1} + K_1 \cdot e_t$

$\hat{W}_{t|t} = \hat{W}_{t|t-1} + K_2 \cdot e_t$

filtrado de varianza

$P_{1,t|t} = P_{1,t|t-1} - K_1 \cdot L'_1$

$P_{2,t|t} = P_{2,t|t-1} - K_1 \cdot L'_2$

$P_{4,t|t} = P_{4,t|t-1} - K_2 \cdot L'_2$

end

Donde $P_{i,t-1|t-1}$ es el bloque i de la matriz de covarianza del filtrado de estado y $P_{i,t|t-1}$ de la predicción.

Capítulo 2

R library: `sm4sd` (*Statistical Modeling for Sensor Data*)

En este capítulo explicamos la librería *Statistical Modeling for Sensor Data* (en abreviatura `sm4sd`) de *R* y los conocimientos previos. La construcción de `sm4sd` es la parte original de esta tesis y se base en los objetos de class `S4`, un sistema orientado a objetos (sistema OO) con formalidad y rigor respecto a `S3`. Vinculado a este capítulo es el conjunto de códigos en R incluyendo la definición de las clases, los algoritmos asociados y sus documentaciones mediante `roxygen2`.

Para llegar a este punto, es necesarios hacer una breve introducción de los conceptos asociados como los diferentes sistemas OO, el motor de los métodos, los posibles repositorios existentes y una explicación de los componentes básicos de un paquete. El objetivo de este capítulo no es sólo explicar los elementos de `sm4sd`, sino también dar un empujo a las personas que querían tener su propio paquete.

2.1. Breve introducción de *R* y reflexión retrospectiva

Según Ocaña-Riola [2017]: La estadística es la parte de la matemática que estudia la variabilidad y el proceso aleatorio que la genera siguiendo leyes de probabilidad. Esta variabilidad puede ser debida al azar, o bien estar producida por causas ajenas a él, correspondiendo al razonamiento estadístico diferenciar entre la variabilidad casual y la variabilidad causal. (p.198) Y la estadística computacional es la interfaz entre la estadística y la ciencia computacional que se ocupa de aplicar los métodos estadísticos a los fenómenos observados para verificar unas hipótesis. Una de las herramientas más usada hoy en día para la comunidad estadística es *R*.

R es una lenguaje de programación y entorno de software libre para la estadística computacional y, además, está dando una base amplia de estadísticos y técnicas de visualización. En el nivel del paradigma de programación, la programción orientada a objetos (OOP: *Object Oriented Programming*) se utiliza en el propio *R*. Es decir, todo en R es un objecto y un objecto es una instancia de una class formada por un

conjunto de variables o funciones para un determinado tema. Actualmente, hay varios sistemas OO implementados en esta lenguage:

1. `Base Types` (tipos base), una estructura de *C* que describe la topología de cada objeto en la memoria. Esta estructura incluye el contenido y el tipo del objeto y, también, las informaciones necesarias para manejar la memoria. En *R* hay un total de 25 tipos base, por ejemplo `NULL` (`NILSXP`), `logical` (`LGLSXP`)[1] y etc. Los tipos base no forman exactamente un sistema de OO, porque sólo el *R-Core* puede crea nuevos tipos.
2. `S3`, el primer y más simple sistema de OO en *R* con una alta ocupación en el repositorio `CRAN`. Asimismo, éste es el único sistema usado en los paquetes `base` y `stats`. Este sistema goberna la *R* la manera de manejar los objetos de diferentes clases. Estas manipulaciones se realizan a través de unas funciones llamadas genéricas y este tipo de funciones posee los métodos (o algoritmos) `S3`. Este sistema no tiene una definición formal de clases y tampoco require muchos conocimientos en programción.
3. `S4` tiene un funcionamiento similar a `S3`, pero añadiendo la formalidad y el rigor. Diferente al anterior, en este sistema hay definición formal de clases que proporcionan descripción y representación de ellas. Los métodos aún pertenecen a las genéricas y no las clases, pero con la posibilidad de dominarlos a través de las genéricas definidas, por una función auxiliar, basando en la adición de argumento, nombre de clase.
4. `RC`, las clases de referencias son un nuevo sistema OO en *R*. Este sistema se diferencia a los dos anteriores en dos maneras. Primeramente, los métodos de `RC` pertenecen a los objectos y no las genéricas. Además, los objectos son mutable. Este tipo de clases no interviene a la construcción de `sm4sd` y, por este motivo, no se trata en esta tesis. Si alguien le interesa, puede encontar informaciones en el libro Advanced R[2].

Un de los elementos mencioanados en la introducción de los diferentes sistemas de OO es la clase de funciones genéricas (polimorfismo). Según su documentación (`?"genericFunction-class"`), las funciones genéricas son objetos de funciones extendidas, que contienen informaciones utilizadas en la creación y el envío de método para esta función. Es decir, las genéricas comportan como un centro de control que envía el método correcto dependiendo la clase de entrada. El método de envío consiste en unos pasos ordenados:

1. mira la clase de primer argumento;
2. concatena la función y el nombre de clase separando por un punto;
3. si la función anterior existe, aplica esta función concatenada con los mismos argumentos mediante una función `call`;
4. en caso contrario, concatena la función original con `default` y llamar la;
5. y, finalmente, devuelve un error si no se encuentra esta función.

[1] El nombre en las paréntesis es su nombre correspondiente en la lenguaje *C*, *C type names*.
[2] `http://adv-r.had.co.nz/OO-essentials.html`

En caso concreto, ¿cómo actua la función `print` a un objeto de `data.frame`? La `print` asímima ya es una genérica y para verificarlo podemos usar la función `isGeneric`. Adicionalmente, es posible ver los métodos definidos de esta función para distintos objetos mediante `methods`[3]:

```
[1] "print.crayon"     "print.data.frame" "print.Date"       "print.default"
```

En la realidad, estamos aplicando el método `print.data.frame`, pero la genérica `print` se encarga de encontrarla.

Otro elemento mencionado es repositorio y un repositorio en *R* es un almacén o lugar donde se depositan los paquetes (*Packages*). Actualmente, existen dos repositorios principales: `CRAN` y `BioConductor`. `CRAN` (*Comprehensive R Archive Network*), como indica su nombre, es una red de servidor ftp y web que almacenan versiones idénticas y actualizaciones de código y documentación para *R*. Para tener un paquete en `CRAN` hay que pasar dos procesos, la creación correcta de un paquete y la validación automática por `CRAN`. Sin embargo, en estos dos procesos *R* solo detecta los errores sintácticos y no revisa los código. Ésto es justa la diferencia principal de estos dos repositorios. `BioConductor` proporciona herramientas para el análisis y la comprensión de datos genómicos de alto rendimiento. Tods los autores de los paquetes en este repositorio tienen que registrar en su WEB y todos los paquetes sometidos serán revisados por un técnico. En resumen, las ventajas de usar el repositorio `BioConductor` son

1. Area de foco, `BioConductor` enfoque el tema de bioinformática y `CRAN` es un repositorio general.
2. Calidad, cada package en `BioConductor` es probado en todos los principales sistemas operativos como Linux, Mac y Windows.
3. Compatibilidad, todos los paquetes tiene una misma estructura de datos y una paquete base común. De esta forma, se facilita el uso cruzado de varios paquetes.

Sin embargo, `sm4sd` no se trata de este área y la única opción es `CRAN`.

Evolución de libreria y reflexión retrospectiva Antes de realizar esta tesis casi no tenía ningua experiencia en el desarrollo en el entorno *R* u otros plataformas, sino como un usuario final. Actuar como un desarrollador o un usario son dos juegos distintos. Es bastante fácil de utilizar las librerias existentes a un problema concreto y, durante este proceso, podemos ignorar todos los pasos intermedios. Sin embargo, crear una desde cero o basando en algunas bases resulta difícil pero posible. En mi caso, el pilar de esta libreria es el análisis del DLM a la conductiviad eléctrica elaboró por **Jose A. Sanchez-Espigares**, mi tutor de esta tesis y el coautor del artículo de Basem et al. [2018]. Esto es solamente el inicio de una aventura.

Para reproducir los resultados sin conocimientos ningunos de DLM, tuve obligarme a hacer notas sobre los códigos incluso el pseudocódigo basando las salidas. Luego, con las ayudas de la libreria `dlm` tampoco pude entender completamente el

[3]Para simplificar el output, ya he ubicado directamente su algoritmo correspondiente.

procedimiento hasta que encontré la viñeta y algunos articulos técnico del mismo autor, Giovanni Petris. La parte dificultosa de `dlm` es la definición de los coeficientes que varian respecto al tiempo o incluir una extra variabilidad en algunos instantes. Por ejemplo, el riego de los cultivos es una acción normal, pero no es tan normal para el análisis. Un aumento del nivel de agua reduce la concentración de sal y provoca una caida o subida enorme de los indicadores. Si no tomamos acciones, las observaciones afectadas se convierten a unos outliers que impiden la validación del modelo. Actualmenete, el modelo de DLM definido en esta libreria toma en cuenta estos eventos exteriores y realiza una búsque de la ventana de afectación correspondiente.

Cuando salté este valle, los problemas vinieron uno tras otro. ¿Cómo crea una libreria y cómo son las clases `S4`? Unas de las referencias usadas en estos dos temas y las considero útiles son *Instructions for Creating Your Own R Package - MIT*[4] y *A (Not So) Short Introduction to S4*[5]. Crear un proyecto de *Package* en `Rstudio` no es misterior y solo hay que seguir los pasos detallados en la primera referencia anterior. El gran problema vino un poco después, una de las propiedades de `S4` es la herencia: las clases pueden herendar las propiedades o representaciones a su clase padre. Entonces, antes de progrmar es necesario construir las relaciones de dependiencia y representaciones adicionales de estas clases según el enfoque. Ahora misma, están definidos tres tipos de clase: la clase base (datos), el conjunto de modelos (DLM y el modelo Hilhorst) y la red recurrente. Los últimos dos tipos de clase son extensiones de la primera. Si el objetivo es analizar los datos brutos, la linea de seguimiento es el conjunto de modelos y la red recurrente sirve para hacer simulación. En la sección **S4**, hablaremos la ventaja de `S4` y la construcción y la utilidad de estas clases con más detalles.

El camino hacia al éxito nunca es una línea recta. Durante el proceso de programación, encontré probelmas concretos como el uso de algunas funcionas desconocidas, algunas mejoras o etc. Para enfrontar estas dudas, las herramientas que utilicé fueron *Stack Overflow*, *Stack Exchange* u otros webs similares. Sin embargo, la forma de consulta más fácil es el buscador *Google*.

Una vez tener todas las clases y los métodos listos, el último reto fue la documentación. Para mí, este fue realmente un problema mayor debido a una capacidad limitada de redactar en inglés. La documentación siempre es una tarea densa pero resulta útil. Una buena descripción de los objetos hacen que el usuario final se entienda la finalidad de cada uno y los utilizan correctamente. Para hacer estas ayudas hay dos maneras: crear directamente los archivos de ayuda donde describen el objeto o usar el paquete auxiliar `roxygen2`. El la próxima sección, exploraremos los componentes básicos de un paquete y el uso de `roxygen2`.

2.2. Package y roxygen2

Un paquete en *R* es la unidad fundamental, que incluye códigos, datos, documentaciones y pruebas (en algunos casos), con el objetivo de compartir. Una libreria es

[4] http://web.mit.edu/insong/www/pdf/rpackage_instructions.pdf
[5] http://christophe.genolini.free.fr/webTutorial/S4tutorialV0-5en.pdf

simplemente un directorio que contiene un paquete instalado. Antes de crear un proyecto de `package` en *Rstudio*, hay que tener un nombre para el paquete, un nombre único que evoca el problema o forma por abreviaturas adecuadas. Como dijo Phi Karlon, solo hay dos cosas difíciles en informáticas: invalidación de caché y nombrar cosas. Estoy totalmente de acuerdo con Phi, este paquete no tiene su nombre propio hasta la última etapa de la tesis.

Una vez tener un proyecto nuevo, se crea automaticamente un directorio con los componentes esenciales:

1. `R/`, un subdirectorio contiene los algoritmos o códigos;
2. `man/`, contiene documentaciones de los anteriores;
3. `DESCRIPTION`, la metadata sobre el paquete;
4. `NAMESPACE`, un espacio que puede contener la importación de funciones exteriores o la exportación de objetos;
5. y `xxx.Rproj`: el inicio del proyecto.

La distribución de estos componentes al proyecto `sm4sd` se muestra en la figura siguiente:

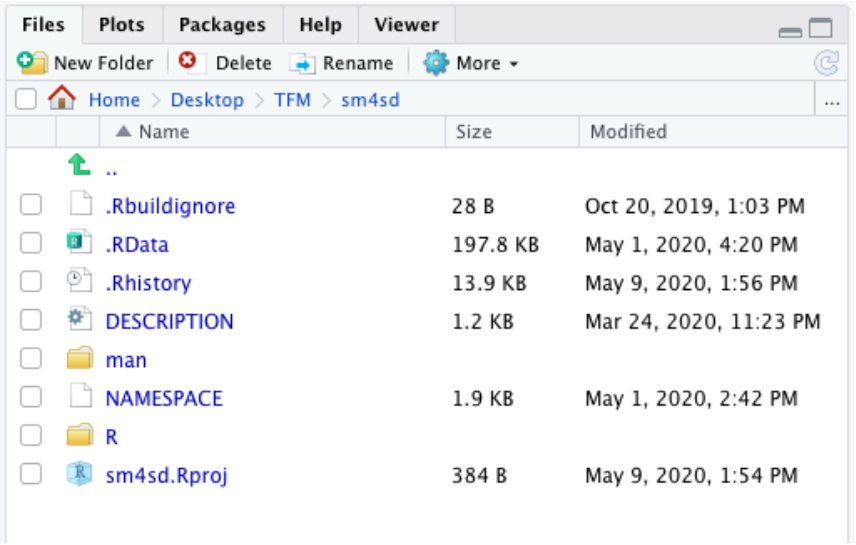


Figura 2.1: EL directorio del proyecto ‘sm4sd‘

Todos los códigos tanto la definición de clases o los algoritmos correspondientes se encuentra en el sub-directorio `R/`. No existen misterios en esta parte, pero siempre es mejor organizar y nombrar los archivos `.R` de una forma más correcta. Por ejemplo, agrupar los algorimos por la utilidad o semejanza y nombrar el archivo con un nombre que ataca el problema. La `DESCRIPTION` contiene una serie de informaciones sobre el paquete:

1. El título y una breve descripción que indica el área de aplicación.
2. Los autores y el contacto de los mantenedores. En muchos casos, los autores o alguno de ellos no mantienen el paquete.

3. Una lista de los paquetes que contienen funciones exteriores usadas en los algoritmos propios. Hay 3 forma de listar estos paquetes que son `depend`, `import` y `suggest`.
   - `depend`: en esta sección podemos listar el software o los paquetes necesarios para una funcionalidad correcta.
   - `import` y `suggest` son muy parecidos y la única diferencia es la cortesia. Los paquetes listados en `import` se instalan automaticamente en la instalación del propio `package`. Esto no ocurre en el caso de `suggest` y, por tanto, antes de usar una determinada función hay que verificar la existencia de su paquete.
4. `Collate` controla el order de lectura de los archivos `.R`. Cuando construimos un paquete, el orden de los `.R` se lean por su orden alfabético, pero ¿qué pasa si alguno de ello depende del objecto que viene un poco después, como el caso de las class `S4`? En estos casos, tenemos que especificar al constructor de paquete que lea el primero el archivo de dependencia. Para indicar un orden correcto es muy fácil de utilizar `Roxygen2`.

`Roxygen2` es una librería creada por Hadley Wickham, un de los autores principales y también el autor de `ggplot2`, con el objetivo de describir las funciones o los objetos en comentarios especiales para generar posteriormente los doumentos `.Rd` (las ayudas), `NAMESPACE` y, si es necesarios, `Collate` en `DESCRIPTION`. Una ayuda simple contiene al menos los siguientes elementos:

1. `\name{name}`, el nombre del objecto o la función;
2. `\alias{topic}`, una sección que especifica todos los temas que se trata del objeto descrito;
3. `\title{Title}`, la información de título con la primera letra mayúscula;
4. `\description{...}`, una corta descripción sobre la funcionalidad del objecto;
5. `\usage{fun(arg1, arg2, ...)}`, la llamada de una determinada función con sus argumentos, para las classes `S4` no tienen este apartado;
6. `\arguments{...}` o `\slot{...}` (para la clase de `S4`), una descripción de los parámetros de una función o las propiedades de una clase;
7. y `\details{...}` si es necesario, una explicación detallada del objeto.

Redactar estos archivos RD manualmente es una tarea costosa, pero con la ayuda de `Roxygen2` se puede simplificar.

El trozo de códigos siguiente es la documentación (una versión ilustrativa) de una clase de `S4` que representa el modelo de Hilhorst. Un modelo de Hilhorst contiene informaciones de la conductividad eléctrica del suelo y la permitividad dieléctrica relativa del agua y del suelo (ver Eq 1.17) con un offset prefijado.

```
#' Class SM.HL --- An S4 class to represent the Hilhorst model.
#'
#' @slot call An object of class \code{\link[base]{call}}
#'  returning an unevaluated function call.
```

```
#' @slot offset A no negative numeric value indicating
#'  $\epsilon_{\sigma_b= 0}}$.
#'
#' @exportClass SM.HL
#' @export SM.HL
SM.HL <- setClass(Class = "SM.HL",
                  slot = list(
                    call = "call",
                    offset = "numeric"
                  ),
                  contains = "SM"
)
```

Las informaciones del suelo y del agua están incluidas en una clase padre cuyo nombre `SM`[6]. Por tanto, las únicas variables (o *slots*) hay que documentar son `call` y `offset`. Para documentar estos slots utilizamos unos comandos de `roxygen2` que empieza por el etiqueta (*tag*) `"@"`. Además, para diferenciar los comentaciones normal y de `roxygen2` se utiliza `"#'"`. Posteriormente, el propio núcleo se encarga de traducir estos comentarios a un archivo RD.

Los tiquetas más usados a lo largo de este trabajo son `@slot`, `@param` (el argumento de una función), `@section` (crear una nueva section), `@note` (hacer una nota), `@examples` (ejemplos del objeto), `@import` (importa de una libreria entra), `@importFrom` (importar una función de un determinado paquete) , `@export` (exportar una función), `@exportClass` (exportar una clase) y `@exportMethod` (exportar un método de una clase). El uso de las etiquetas relacionadas con `import` indica a la función genérica el método requerido en los algoritmos. Estos algoritmos no serán visibles a los usuarios si no los exportan.

Respecto a la estructura, las diferentes áreas de un Rd están separadas por una línea vacia y la primera indica el título del objeto. Si no existe una redacción explícita de descripción, entonces este área será mismo que el título. Utilizamos el etiqueta `@section` para crear una nueva section, por ejemplo la sección de detalles. Es difícil de introducir todos los posibles usos de esta libreria, siempre podemos encontar más informaciones en su documentación

```
help(package == "roxygen2")
```

2.3. Sistema S4

`S4` es un sistema más estricto en comparación con `S3` que tiene una definición formal de las clases y esta formalidad hace que las propiedades de OOP se transfieren a `S4`. Estas propiedades son el encapsulamiento, la herencia, el polimorfismo y la

[6] En este caso, esta clase es una clase virtual extendida de `RS`. En la proxima sección, enfocaremos estas clases.

abstracción. La abstracción significa el uso de una estructura simple a representar una tarea compleja. Por ejemplo, sabemos crear un vector o un *dataframe* pero no conocemos los detalles. Esta propiedad no es una propiedad siginificativa de las clases `S4` y no discutimos en esta sección.

Encapsulamiento El encapsulamiento es el mecanismo de proteger a las informaciones de los métodos exteriores, es decir, las únicas maneras de acceder o manipular estas informaciones son los métodos internos. En otros entornos como *PYTHON*, la definición de la clase y los métodos disponibles forman un único objecto. Sin embargo, en el caso de *R* las clases y los métodos de `S4` se crean separadamente utilizando funciones en el paquete `methods`, `setClass` y `setMethod`.

Algunos de los argumentso de `setClass` (`?setClass`) son `Class`, `representation` o `slots` y `prototype`, donde `Class` indica el nombre de clase que queriamos definir, `representation` o `slots` contiene las informaciones de una clase con un tipo predeterminado y `prototype` es la inicialización de estas informaciones. Existe una pequeña diferencia entre `representation` y `slots`, la primera puede contener definición de la clase padre, en el caso de herencia, y la segunda solo indica las informaciones adicionales y el control de herencia se hace mediante un argumento adicional `contains` donde hablaremos luego. Es más recomendable de usar `slots` y `contains` en lugar de `representation` para tener menos confunsión a los usarios. Estos componentes forman una definición básica de una clase `S4`.

El objetivo de la librería `sm4sd` es estimar la conductividad eléctrica del agua en el suelo basando a las señales medibles y la clase base de todas aquelles definidas en esta librería es `RS`, que contiene las señales brutas.

```
RS <- setClass(Class = "RS",
               ## variables
               slot =  list(
                 sigb = "numeric",
                 epsb = "numeric",
                 epsp = "numeric"
               ),
               ## prototype: initiation of the class
               prototype = prototype(
                 sigb = numeric(0),
                 epsb = numeric(0),
                 epsp = numeric(0)
               )
)
```

El bloque de códigos anterior representa la definición de una clase cuyo nombre `RS` (*Raw Signals*) con tres variables numéricas representadas a σ_b, ϵ_b y ϵ_p inicializadas por un vector de tamaño cero. La ventaja de especificar el tipo de las variables es la validación posterior de la creación de una instancia de `RS`. Devolverá un error cuando

el tipo de algunas variables no sea deseable por la definición. A parte de la restricción del tipo de variable es posible añadir más restricciones usando `setValidity` (`?setValidity`). Para construir los modelos requerimos que la longuitud de estas variables sea la misma y la adición de esta validación se logra por el siguiente bloque:

```
.RS.setvalidation <- function(object){
  n1 <- length(object@sigb)
  n2 <- length(object@epsb)
  n3 <- length(object@epsp)

  cat("--- Validating the RawSignal class ---\n")
  if( (n1 == n2) & (n1 == n3) ) {
    cat("OK\n")
    TRUE
  }else {
    cat("Inputs must have same length!\n")
    FALSE
  }
}
setValidity("RS", .RS.setvalidation)
```

Esta propiedad no sólo aparece en la creación de una instancia, sino también en los métodos. Los métodos asociados de una clase se crean por `setMethod` especificando la asignatura y, a parte de este argumento, esta función incluye el nombre de la función genérica y la definición del método concreto. En el `S4`, para acceder los valores de una determinada variable se utiliza el símbolo `@`. Sin emabrgo, para evitar el uso directo de este operador suele utilizar una envoltura mediante la función genérica `"["`, extraer partes de un objeto.

```
.RS.get <- function(x,i,j,drop){
  switch(EXPR = i,
         sigb = return(x@sigb),
         epsb = return(x@epsb),
         epsp = return(x@epsp),
         stop("Error:",i,"is not a RS slot")
  )
}
setMethod(f = "[",
          signature = c("RS", "character", "missing", "missing"),
          definition = .RS.get)
```

La función genérica `"["` contiene cuatro argumentos que son `x`, `i`, `j` y `drop` y para mantener su consistencia en la definición del método, en este caso `.RS.get`, debe incluyerlos. La asignación de un método a una clase determina se realiza por los argumentos `signature` y `definition`. La signatura es un vector que especifica la clase

de los argumentos de un método definido. En general, el primero argumento de un método especifica el objeto que quiere realizar una cierta operación y para `setMethod` indica la pertenencia del método. En el caso de que alguno de los argumentos no ha sido utilizado en la definición debe ser marcado por `missing`.

Si queremos asignar un nuevo método a una clase asociando una función genérica no definida, debemos crear previamente la dicha genérica con `setGeneric`. Algunos ejemplos se muestran en la sección de los métodos de `sm4sd`.

Herencia Gracias a la herencia, es posible de crear una nueva clase que extiende de una antigua clase denotada por la clase padre. Es decir, crear una nueva clase que contiene todas las informaciones de la clase padre, más todas las nuevas informaciones si fueran necesarias. En este caso, la clase hija no sólo herienda las informaciones, sino también los métodos definidos a la clase padre y su validación.

Volvemos a la nuestra librería, las líneas de investigación de las señales brutas son la construcción de los modelos para analizar la conductividad y la simulación mediante los modelos de red neuronal recurrente. Actualmente, existe dos modelos distintos para el análisis, el modelo determístico de Hilhorst y el modelo estádistico, dlm. Para marca la diferencia entre estas dos líneas y tener un mismo orígen de estos dos modelos, una clase extendida a `RS` que marca la línea del modelo es necesaria.

```
setClass(Class = "SM",
         contains = c("RS",
                      "VIRTUAL")
)
```

El argumento `contains` indica las clases de herencias y, en este caso, creamos una clase `SM` (*signal model*) que contiene la clase `RS` y `VITRUAL`. Si la class es vitural o la contiene, un intento a generar una instancia mediante el constructor o la función `new` devolverá un resultado con errores. Asimismo, la clase `SM` representa una clase virtual que contiene todas las informaciones de `RS` y no es posible de crear una instancia concreta. Podemos obtener la definición de clase mediante la función `getClass`:

```
Virtual Class "SM" [in ".GlobalEnv"]

Slots:

Name:     sigb    epsb    epsp
Class: numeric numeric numeric

Extends: "RS"
```

Los métodos funcionan de alguna manera similar. Imaginamos que tenemos un método, `foo`, definido a la clase `RS` pero no está definida `SM` (supongamos que esta clase no es una clase virtual). Cuando aplicamos `foo` a `SM`, la función genérica se encarga de verificar la existencia del determinado método para `SM`. En el caso negativo, se halla el mismo método para su clase padre y se lo aplica usando `callNextMethod`. De esta forma, podemos ampliar un determinado método para las clases extendidas. Para ver el ejemplo de este uso, se puede consultar el código fuente del método `show` en el apéndice.

Polimorfismo Según el glosario de Bjarne Stroustrup [2012] de los términos *C++*, un polimorfismo proporciona una única interfaz a entidades de diferentes tipos. Tanto en el `S3` como en el `S4`, una única función puede tener diferentes formas según la clase. Una función genérica en `S4` opera de una manera similar a `S3`, pero incluye la herencia de los métodos.

2.4. Clases y métodos en `sm4sd`

En esta sección, describimos todas las clases y los métodos definidos en la librería `sm4sd`. En la sección de clases, proporcionamos la definición y la utilidad de cada una junto con los métodos disponibles. Para ver el detalle de los métodos, se dirige a la sección posterior donde los explicamos detalladamente según la genérica. Los códigos de los métodos se encuentran en el apéndice A.

2.4.1. Clases

Como mencionado anteriormente, existe una bifurcación sobre la utilidad de las señales brutras, modelización y simulación. Si iremos más profundo a la modelización de los datos, habrá otra bifurcación en esta rama. Hasta momento hay un total de 6 clases en la librería que son:

Clase RS Esta clase ha sido introducido como el ejemplo de creación de `S4` que contiene únicamente las señales brutas con el objetivo de extender. Los métodos disponibles son `show`, `print`, `get` o (`"["`), `plot`, `plotCor`, `buildClass` y `buildRNN`.

Clase SM La clase `SM` es una clase virtual que extiende directamente a la clase `RS` sin informaciones adicionales y representa la clase raíz de la modelización. A partir de esta clase virtual se extienden todas las nuevas clases (los modelos distintos) con el fin de analizar y estimar la conductividad eléctrica del agua en el suelo. Según las literaturas existentes, hay únicamente dos métodos que son Hilhorst y DLM. El primero es el método que está usando en los dispositivos de tecnología sensorial con un offset fijo donde requiere un estudio previo de él. Este modelo es determinísitico y el resultado depende mucho el valor de offset. El segundo es un modelo estocástico y su ventaja es modelizar conjuntamente la conductividad y el valor de offset. De esta forma incluimos un ruido aleatorio.

Clase SM.dlm La clase `SM.dlm` es la extensión directa de `SM` que contiene las definiciones previas de un DLM. Esta informaciones adicionales cubren varios aspectos:

1. el componente estacional de los datos brutos,
2. los eventos incontrolables o las intervenciones humanas junto con el máximo retraso de estos efectos
3. y, finalmente, dos indicadores relacionados a la validación y el proceso de paralelización.

En la serie temporal, los datos pueden presentar algunos patrones que repiten cada cierto tiempo. El tiempo que transcurre se denota por la frecuencia o ciclo y está relacionado al componente estacional. Por ejemplo, la temperatura ambiental de un medio plazo puede tener un ciclo diario y si recogemos las informaciones cada hora, la frecuencia correspondiente será 24 horas.

Para modelizar correctamente el fenómeno observado, la inclusión del componente temporal no será suficiente. En muchos casos, se registran valores atípicos en los datos debido a un evento incontrolable o intervención humana, la lluvia o el riego en el nuestro caso. Gracias a DLM, se puede introducir una extra variabilidad a los instantes que producen comportamiento estraño debido a los anteriores. De esta forma, se puede validar correctamente el modelo una vez ajustado. Sin embargo, el efecto de estos factores exteriores pueden diferir según los casos. Para hallar el efecto conjunto se aplica una búsqueda de modelos basando la máxima verosimilitud. Es decir, dando un retraso máximo, se construyen los modelos con todas las posibles combinaciones de retraso a cada factor y el mejor modelo es aquello que tiene la mejor estimación de máxima verosimilitud.

En general, un número exceso de modelos pueden causar problemas computacionales. El número de modelos construidos se determina por el retraso máximo elevado al número de eventos exteriores. Para cuatros eventos y máximo retraso de cinco, el número de modelo sera $5^4 = 625$. Para evitar este problema, la asignación de validación se requiere en la creación de la clase. Un número mayor que 200 con `verify == TRUE` devolverá un error en el proceso de validación. Finalmente, el argumento `parallel` controla el uso de paralelización en la búsqueda automática que será implementado en la próxima versión.

En el bloque siguiente se presenta la definición de la clase `SM.dlm` y su validación.

```
SM.dlm <- setClass(Class = "SM.dlm",
                   slot = list(
                     call = "call", freq = "numeric",
                     indicators = "list", lagMax = "numeric",
                     verify = "logical", parallel = "logical"
                   ),
                   contains = "SM"
)

.SM.dlm.setvalidation <- function(object){
```

```
  freq <- object@freq
  index <- object@indicators
  lagMax <- object@lagMax
  verify <- object@verify
  cat("--- Validating the SignalModel \'dlm\' class ---\n")
  if(freq < 1 | floor(freq) != freq ){
    stop("The frequency must be POSITIVE INTEGER.")
  }

  if(length(index) != 0){
    events <- unlist(index)
    events.round <- round(events)
    aux_count <- sum(events <= 0)
    aux_equal <- all.equal(events, events.round)

    if( aux_count != 0 | !aux_equal){
      stop("Event indicators must be POSITVE INTEGER.")
    }
    if( lagMax <= 0 | floor(lagMax) != lagMax){
      stop("Maxim lag must be a positive integer.")
    }
    if(lagMax ** length(events) > 200 & verify){
      stop("The number of model for gridsearch is excced to 200.")
    }
  }

  cat("OK\n")
  TRUE
}
setValidity("SM.dlm", .SM.dlm.setvalidation)
```

Los métodos disponibles para esta clase son `show`, `print`, `get` o `"["`, `fit` y `extractMeasures`.

Clase SM.dlm.fitted En la presente versión de `sm4sd`, se considera el offset constante y los únicos parámetros fijos del modelo son V y W que representan la varianza de la observación y el producto de σ_b y ϵ_p. Además, si existen factores exteriores, se estima una varianza por las categorias. Teniendo en cuenta que los eventos de la misma categoria pueden tener una influencia distinta, la próxima versión dará opción de estimar esta varianza por indicador o por categoría.

La clase `SM.dlm.fitted` se crea aplicando el método `fit` a la clase `SM.dlm`. Una vez entrenado el modelo, se añade las informaciones addicionales:

1. `parameters`, los parámetros del mejor modelo;
2. `filtered`, una lista que contiene los valores esperado de filtrado;

3. `smoothed`, una lista que contiene los valores esperado de alizado;
4. `lags`, retraso de los indicadores para el mejor modelo estimado;
5. `seasonalSign`, el análisis del componente estacional se aplica en el caso de que la frecuencia sea mayor que una;
6. y `tracking`, el seguimiento de la historia de búsqueda ordenado por el redimiento.

Los métodos disponibles son `show`, `print`, `get`. `getMod`, `residualDiag` y `extractMeasures`. Y la definición de clase se encuentra en el bloque siguiente.

```
SM.dlm.fitted <- setClass(Class = "SM.dlm.fitted",
                          slot = list(
                            parameters = "list", filtered = "list",
                            smoothed = "list", lags = "list",
                            seasonalSign = "character",
                            tracking = "data.frame"
                          ),
                          contains = "SM.dlm"
)
```

Clase SM.HL La clase `SM.HL` es la extención directa de `SM` que representa el modelo de Hilhorst. Comparando a la clase DLM, esta clase es simple por ser un modelo determinístico y la única información adicional es un offset no negativo. Para extraer las medidas de conductividad se utiliza el método `extractMeasures`.

```
SM.HL <- setClass(Class = "SM.HL",
                  slot = list(
                    call = "call",
                    offset = "numeric"
                  ),
                  contains = "SM"
)

.SM.HL.setvalidation <- function(object){

  offset <- object@offset
  cat("--- Validating the Magnus Persson class ---\n")
  if(  length(offset) != 1 | offset < 0 ){
    cat("he offset should be no negative length one numeric variable.")
    FALSE
  }else{
    cat("OK\n")
    TRUE
  }
```

```
}
setValidity("SM.HL", .SM.HL.setvalidation)
```

Clase SM.rnn Comparando a las clases anteriores, `SM.rnn` presenta algunos comportamientos distintos a las anteriores aunque es una extensión de `SM`. Debido a la incompatibilidad de los objetos `Keras` en `S4`, no es posible de tener una red en el *slot*. Una de las posibles soluciones es guardar el modelo entrenado en el directorio local y sólo conservan las informaciones de ajuste. Casi todas las informaciones en esta clase son los argumentos del método `buildRNN` excepto la historia de busqueda.

```
SM.rnn <- setClass(Class = "SM.rnn",
                   slot = list(
                     call = "call", model_path = "character",
                     compile_options = "list",
                     optim_hyperparameters = "list",
                     preprocess_parameter = "list",
                     tracking = "data.frame"
                   ),
                   contains = "SM"
)
```

Para esta clase están disponible el método `show` y la función `RNN_predict`.

2.4.2. Métodos

En la primera versión de la libería hay un total de once métodos de `S4` y una función adicional para la clase `SM.rnn`. Cuatro de ellos tienen la genérica definida en *R* que son:

1. `show`: exhibir la clase de una forma reducida;
2. `print`: mostrarla con más detalles;
3. `[`: extraer la información de una clase;
4. y `plot`: pintar las señales brutas en la misma figura.

Para los restos métodos se ordenan por su utilidad:

plotCor(x, groups, ...) El método `plotCor` elabora una gráfica de correlación entre x_t (el producto de ϵ_p y σ_b) y y_t (ϵ_b) según el número de grupos dado. El objetivo de este método es verificar la relación lineal entre estas dos variables como mostró Hilhorst. Los argumentos son:

- `x`: el objeto de la clase `RS`,
- `groups`: un entero positivo indicando el número de subconjunto de misma longuitud con el order preservado.
- y `...`: los arumentos adicionales para la función `plot`.

buildClass(object, method, freq, ind, lagMax, verify, parallel) El objetivo de este método es construir las clases `SM.dlm` y `SM.HL` a partir de `RS` añadiendo informaciones extras descritas en la definición de cada una. Los argumentos son:

- `object`: un objeto de la clase `RS`,
- `method`: indica el método de extensión aplicada que puede ser *dlm* (`SM.dlm`) o *hl* (`SM.HL`),
- `ind`: para el `SM.dlm`, este argumento refiere a la lista de los eventos exteriores y para `SM.HL` el valor de offser,
- `freq`, `lagMax`: la frecuencia de la serie y el máximo restraso para los factores;
- `verify` y `parellel`: un valor lógico que indica la activación de validación y el proceso de paralelización.

buildRNN(object, compile_options, hyperparameters, preprocess_method, lag, model_path) `BuildRNN` es un método complejo que cosistene en varios pasos: definir los parámetros de un modelo `Keras`, preprocesar los datos según el método y la ventana de retraso, expandir las posibles combinaciones de parametros si alguno de ellos presenta multiple opciones, realizar una búsqueda de modelos y, finalmente, almacenar el mejor modelo en la directorio indicado.

Un modelo `Keras` require unas opciones de compilación como el optimizador, la función de pérdidad y la métrica. También hay que dar los hyperparámetros como el tamaño de lote y el número de *epochs*. Adicionalmente, el modelo definido en `sm4sd` acepta la típica técnica de validación de un modelo *Machine Learning*, la división de *Traning-Test-Validation*. Si algunas opciones no presentan en los argumentos, el proceso de definir los parámetros nos ayuda a verificarlas y asignar el valor por defecto para cada caso.

Por otra parte, una red es sensible a la magnitud de las variables y para tenerlas en un rango similar, las técinicas como la normalización y la transformación *MAX-MIN* son necesarias. En el segundo caso se aplica primero la normalización y luego la transformación. En ambos casos, es necesario de guadar las informaciones de media, varianza, máximo y mínimo para el preproceso de un nuevo conjunto de datos. En una RNN simple con dos variables, el número de parámetros del modelo es escaso y resulta que el componente temporal no se captura bien. Para tener una red más compleja se acepta un argumento `lag` que indica la ventana de retraso. Si este argumento es mayor que una, en el preproceso de datos incluirá una expanción de los datos de entrada.

Un hiperparametro es un parámetro que podemos ajustar dependiendo el redimiento del modelo ajustado. En este caso se acepta multiple opción para realizar una búsqueda de estos hiperparámetros. El mejor modelo RNN se basa el criterio de la suma absoluta de los errores de predición y será guardado en un directorio local. Finalmente, la clase `SM.rnn` se crea con las informaciones anteriores.

Los argumentos son:

- `object`: un objeto de clase `RS`;
- `compile_options`: una lista contiene las opciones de compilación que son optimizador, pérdida y métrica;

- `hyperparametros`: una lista contiene el tamaño de lote, el número de *epochs* y el ratio de test y validación;
- `preprocess_method`: indica el método de preproceso que aplica, *normalized* o *maxmin*;
- `lag`: la ventana de restraso;
- y `model_path`: el directorio donde guardar el modelo.

fit(object) El resultado del método `fit` es una clase `SM.dlm.fiited`. `Fit` es una envoltura de la librería `dlm` que incluye el diseño de un modelo DLM y el entrenamiento de todos los modelos posibles. Si la serie presenta una frecuencia mayor que una, la prueba de χ^2 se aplica al modelo final sin o con el componente estacional para verifica su significación. El único argumento para este método es `object` que indica un objeto de clase `SM.dlm`.

getMod(object, name) En la realidad, la clase `SM.dlm.fitted` no guarda el modelo completo de DLM, sino sus parámetros. Para recuperar el modelo entranado en algunos casos, el método `getMod` nos ayuda a reconstruir un objeto de clase `dlm`. Los argumentos son:

- `object`: un objecto de la librería `sm4sd` y, por momento, solo está diponible para `SM.dlm.fitted`
- y `name`: este argumento no está usado en la presente versión.

residualDiag(object, sd, ask) El diagnóstico de los errores incluye la banda de confianza (1.1.3), el test de χ^2 para los errores estandarizados (1.1.3), qqplot de los errores estandarizados y el test de autocorrelación (1.1.3). Los argumentos son

- `object`: el objeto que require la validación,
- y `ask`, un valor lógica indicando el proceso de una nueva figura.

extractMeasures(object, plot) Una vez ajusta el modelo DLM o pasar el offset al modelo Hilhorst podemos extaer las medidas de conductividad eléctrica de agua. Para realizar esta extracción, este método es necesario. Los argumentos son

- `object`: el objeto que se puede extraer las medidas de la conductividad;
- y `plot`: un valor lógico que indica si se debe trazar el gráfico.

RNN_predict(object, sigb, epsp) `RNN_predict` no es un método de `S4`, sino una función que realiza la simulación. Como su nombre indica, esta función es un algoritmo que predice ϵ_b usando el modelo RNN entrenado. Los argumentos son

- `object`: un objeto de clase `SM.rnn`,
- `sigb` y `epsp` son el vector que contiene las nuevas medidas de la conductividad eléctrica de suelo y la permitividad de agual respectivamente.

Capítulo 3

Aplicación de `sm4sd`

En este capítulo ilustraremos el uso de la librería `sm4sd` al análisis de la conductividad eléctica del agua basando en un conjunto de datos de laboratorio. Este capítulo está organiza en la siguiente manera:

1. La primera sección contiene una pequeña introducción de la extracción de las señales y una exploración de los datos.
2. La segunda sección consiste en la modelización de estos datos mediantes la librería `sm4sd`.
3. En la última sección, haremos un pequeño resumen y posibles mejoras basando el análisis hecho.

3.1. Experimiento y datos

Antes de explorar las señales crudas, revisamos el experimento propuesto por el artículo de Basem et al. [2018] y sus condiciones. En este estudio, se utliza el *Laboratory column experiment* de dos columnas de suelo con una altura de 55 cm proporcionadas por un aspersor, ver la figura 3.1. El límite inferior se controló usando una bomba de vacío a un cabezal de presión constante de -30 hPa. Las columnas fueron empaquetadas con una densidad de 1.4 g/cm^3. El sustrato era arena, el 80 % de la cual era arena fina. El contenido de agua durante el envase fue apriximadamente 4 m^3/m^3. Los sensores de temperaturas del suelo y las reflectometrías de dominio de tiempo (TDR: *Time Domain Reflectometry*) se instalaron en cuatro profundidades: 7, 21, 35 y 48 cm. Según el artículo, la permitividad dieléctrica (ϵ_b), la conductividad eléctrica σ_b y la temperatura del suelo se midieron cada 5 minutos para obtener suficientes observaciones para modelar.

Adicionalmente, se instalaron ventosas porosas para tomar muestras de solución de suelo a diferentes profundidas para validar posteriormente el resultado del modelo. El límite inferior de la columna utiliza una menbrana para drenar el agua. El aspersor está a 5 cm sobre la superficie del suelo y permite que el agua caiga a través de boquillas pequeñas. Se aplicaron cinco eventos de riegos con solución de cloruro de potasio con diferentes conductividades eléctricas. Las columnas estaban libres de sal al principio y antes de comenzaran los eventos de riegos. La temporatura del suelo

fue capturada por termistores y los datos (T_{soil}) se usaron para estimar directamente la permitividad diélectrica relativa del agua de poro del suelo:

$$\epsilon_p = 80{,}3 - 0{,}37 \times (T_{soil} - 20)$$

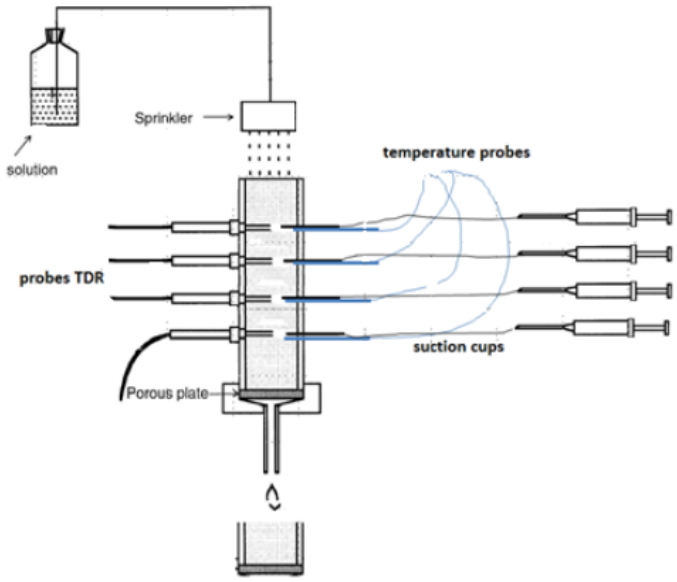


Figura 3.1: Conjunto de Medidas

En este trabajo, solo reproducimos el resultado del conjunto de datos de la primera columna con una profundidad de 21 *cm* y el tercero evento de riego. Con el fin de explorar los datos crudos, aprovechamos los métodos definidos en `sm4sd`. La peculiaridad de los métodos `S4` es que estos métodos no son métodos públicos y solo se puede utilizarlos a través de la clase asociada. La clase base en este paquete es `RS` que contiene las señales crudas. Exsite dos maneras de crear esta clase, el uso del método `new` o el constructor de la clase, normalmente una función cuyo nombre igual al nombre de clase. Dependiendo de la definición de clase en cada paquete, el consutructor puede no existir en algunos casos y, para este paquete, es recomendable de usarlo.

Dando el constructor de la clase `RS`, las series brutas se pasan por argumentos correspondients. Y durante la creación de la instancia, la validación automática se ejecuta y se prueba el tipo y la longuitud de las entradas. Si no hay ninguno problema, la clase se crea correctamente con un breve mensaje de validación:

```
class_base <- RS(sigb = serie$sigb, epsb = serie$epsb, epsp = serie$espp)
```

```
--- Validating the RawSignal class ---
OK
```

Asociado a esta clases, varios métodos nos permite explorar los datos brutos como `show`, `print` y `plot`. Para no mostrar completamente todas las informaciones de las series, usamos el método `show` simplemente llamar el nombre de instancia. Para `RS`, se ilustra la longuitud de las series y los primeros 25 elementos de cada una:

```
class_base
```

```
An object of class RS (RawSignals):

Length = 279 (limited to a 25)

* "sigb" (numeric) :
 [1] 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.14
[16] 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.14

* "epsb" (numeric) :
 [1] 10.2 10.2 10.2 10.2 10.2 10.2 10.2 10.2 10.2 10.2 10.2 10.2 10.3 10.2 10.2
[16] 10.2 10.2 10.2 10.2 10.2 10.2 10.2 10.2 10.2 10.2

* "epsp" (numeric) :
 [1] 78.54 78.52 78.52 78.51 78.50 78.50 78.49 78.48 78.48 78.46 78.46 78.46
[13] 78.46 78.46 78.46 78.46 78.45 78.45 78.44 78.44 78.44 78.44 78.43 78.43
[25] 78.43
```

Igual a otras esturcturas de datos (`data.frame`, `vector`, etc.), el operador `"["` permite extaer completamente un slot y asignar una nueva variable, `sigb <- class_base["sigb"]`, solo indicando el nombre de slot que queriar operar.

En muchas situaciones, hace falta de visualizar los datos para tener un mejor conocimiento de los datos. Por ejemplo, el comportamiento de las series, los cambios que producen los datos atípicos y etc. El método `plot` utiliza `plot.timeSeires` que nos permite representar varias series temporales en una misma gráfica, ver figura 3.2. A parte de la visualización simple de las series, también es interesante comprobar la relación entre ϵ_b y $\epsilon_p \times \sigma_b$. Para esta tarea, dispone el método `plotCor` con un extra arugumento, `groups`, que permite computar la regresión lineal de los diferentes grupos homogenios conservando el orden, ver figura 3.3.

```
plot(class_base)
```

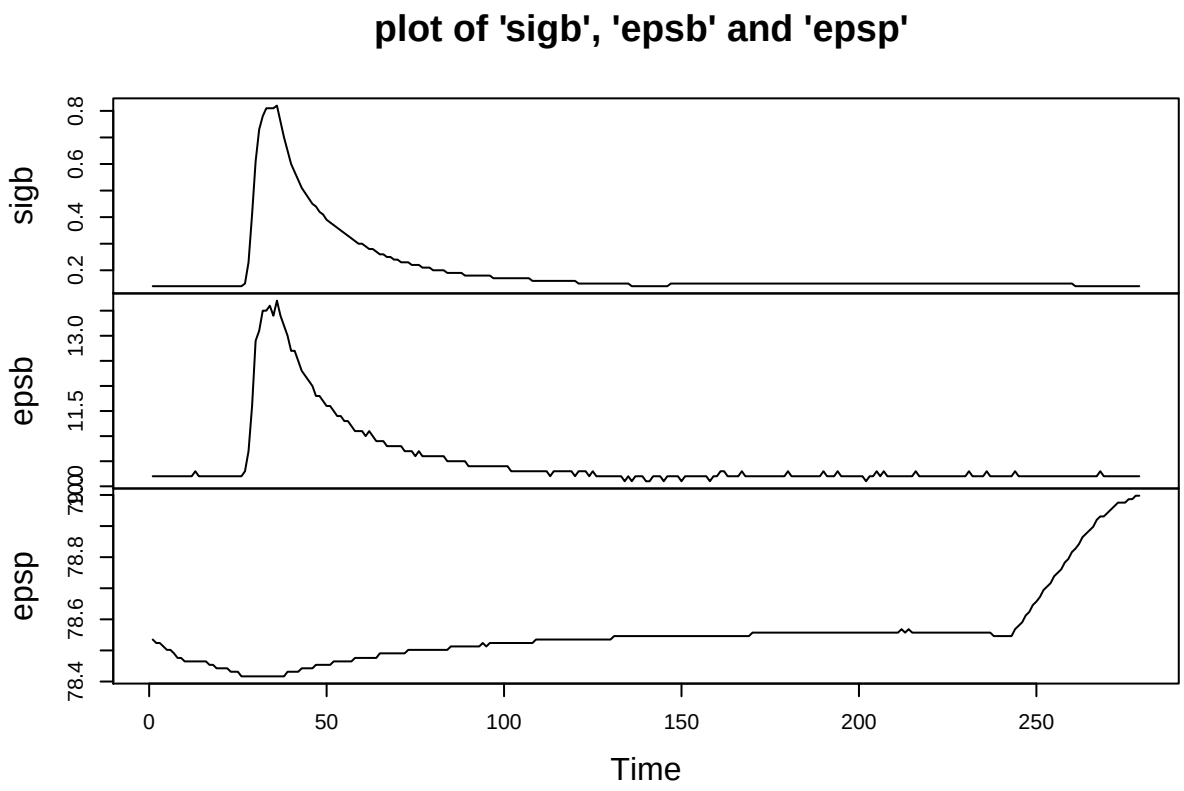


Figura 3.2: Representación gráfica de las señales crudas

```
`geom_smooth()` using formula 'y ~ x'
```

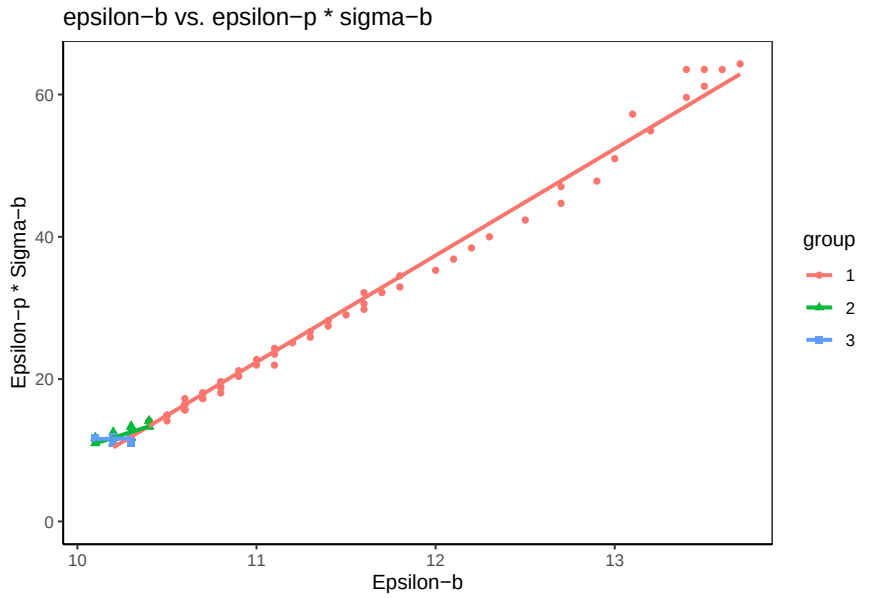


Figura 3.3: Regresión lineal por grupos

Para este conjunto de datos, el comportamiento de la conductividad electrica y la permitividad del suelo, σ_b y ϵ_b, son bastante similares con una magnitud distinta.

Según el artículo, hay tres riegos durante el experimento. Sin embargo, estos tres riegos no están identificados y se observa, mediante la gráfica, un cambio significativo cerca del índice 27. Para una división de tres grupos, aunque el pendiente de cada uno es distinto, se acepta la hipótesis de que la relación es lineal.

3.2. Modelización

El objetivo final del paquete es estimar la conductividad eléctrica del agua. En esta sección, ilustramos dos maneras de hacer estimaciones, DLM y modelo de Hilhorst, y una comparación entre ellas. En el caso de DLM, explicamos adicionalmente el entrenamiento y la validación del modelo.

Construcción y validación del modelo DLM La extendeción la clase `RS` a `SM.dlm` se realiza mediante por el método `buildClass` indicando el `method = "dlm"`. Bansado en las figuras anteriores, es necesario añadir otras informaciones que se utilizarán en la construcción y el entrenamiento del modelo. En este caso, las series no presentan ninguna estructura estacional y un tipo de evento exterior (riego) en el índice 27.

Para indicar los eventos exteriores, se utiliza el argumento `ind`, una lista que contiene diferentes tipos de eventos y los índices correspondientes. Además. estos eventos pueden producir un efecto distinto dependiendo del tipo. Para solucionar este problema, este paquete admite una busqueda del modelos basando en la verosimilitud. Por tanto, en la etapa de definir la clase DLM es necesarios de indicar la máxima ventana de efecto mediante el argumento `lagMax`. En este caso particular, dejamos `lagMax = 20`.

```
freq <- 1
index <- list()
index[[1]] <- c(27)
names(index) <- c("irrigation")
lagMax <- 20
verify <- T
parallel <- F
dlm_base <- buildClass(object = class_base, method = "dlm",
                       freq = freq, ind = index,
                       lagMax = lagMax, verify = verify,
                       parallel = parallel)
dlm_fitted <- fit(object = dlm_base)
```

Una vez extendida la instancia, podemos entrenar el modelo con los ajuste predeterminados mediante el método `fit`. La clase `dlm_fitted` contiene los parámetros del mejor modelo y las restas informaciones importantes:

```
dlm_fitted
```

```
************* Class SM.dlm.fitted, method Show *************

* Call: fit(object = dlm_base)

* Best model parameters :
  ** Estimated parameters of dlm (with exp) :
      V          : 0.00122747017002031
      W          : 4.65850998340902e-07
      irrigation : 15.5739555027348

  ** Associated lag :
      irrigation : 9

* Series without seasonality.

* Index of event indicators :
    irrigation : 27

* Top 5 rows of tracking :

   irrigation    LogLik         V         W irrigation
9           9 -718.5655 -6.702797 -14.57937   2.745601
10         10 -717.6532 -6.701068 -14.56979   2.645004
11         11 -717.6402 -6.692175 -14.69866   2.711576
15         15 -716.9022 -6.701273 -14.71059   2.349515
16         16 -716.8985 -6.695619 -14.80057   2.321483


***************** End Show(SM.dlm.fitted) ******************
```

En el nuestro caso, una ventana de restraso de 9 nos lleva el mejor modelo con una extra varianza 15,57 para el evento de riego. Aunque esta clase no contiene directamente el modelo DLM, si alguien le interesa es posible de extaerlo mediante el método `getMod` y la salida es de la clase *dlm*. No introducimos la estructura de clase `dlm`, pero se puede encontrar el manuel de uso en Petris&An [2010] o mediante la función `str`.

```
dlmMod <- getMod(dlm_fitted)
class(dlmMod)
```

```
[1] "dlm"
```

```
str(dlmMod)
```

```
List of 11
 $ m0 : num [1:2] 0 0
 $ C0 : num [1:2, 1:2] 1e+07 0e+00 0e+00 1e+07
 $ FF : num [1, 1:2] 1 1
 $ V  : num [1, 1] 1
 $ GG : num [1:2, 1:2] 1 0 0 1
 $ W  : num [1:2, 1:2] 0 0 0 0
 $ JFF: num [1, 1:2] 0 3
 $ JV : num [1, 1] 4
 $ JGG: NULL
 $ JW : num [1:2, 1:2] 1 0 0 2
 $ X  : num [1:279, 1:4] 0 0 0 0 0 0 0 0 0 0 ...
 - attr(*, "class")= chr "dlm"
```

Aún nos falta validar el modelo entrenado. Para esta validación, realizamos cuatros pruebas con el residuo del modelo: la banda de confianza, la prueba de χ^2, el `qqplot` y el test de Durbin-Waston. Entre estos pruebas, tres han sido introducidos en la sección 1.1.3 y el `qqplot` sirve para compara los cuantiles teóricos con de la muestra.

```
residualDiag(dlm_fitted, gamma = 0.95)
```

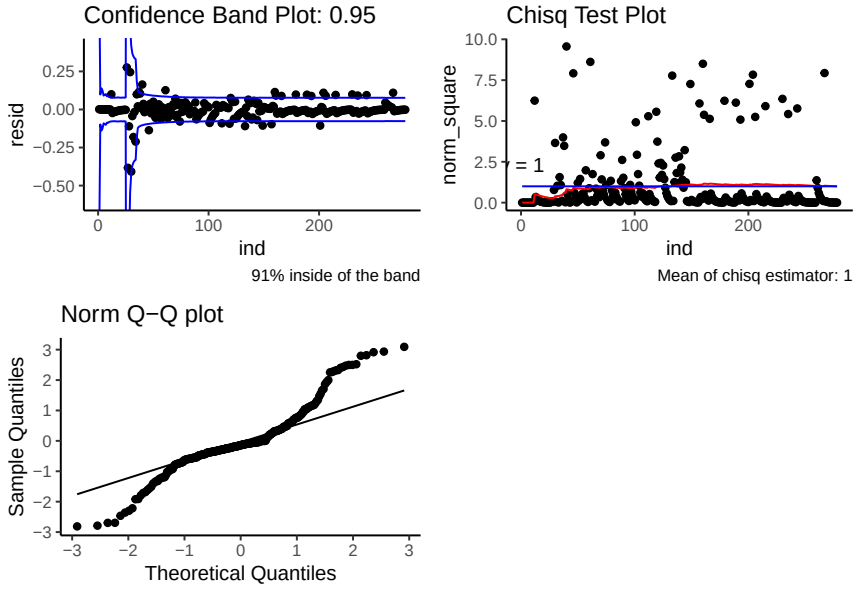


Figura 3.4: Diagnostico de los errores

```
    Durbin-Watson Test : y ~ x

DW = 1.9946, p-value < 2.2e-16
alternative hypotesis: true autocorrelation is greater than 0
```

La intepretación de las gráficas es siguiente:

1. En la primera gráfica, esperamos que 95 % de los erroes caen aleatoriamente en la banda de confianza. Sin embargo, no lo tenemos, 91 % para este caso. Además, algunos de errores en la segunda mitad situan fuera de la banda definida.
2. En la figura de prueba χ^2, la media móvil a largo plazo se aproxima a la unidad mientras que algunos estimadores tienen un valor extremadamente alto. Estos valores extremos pueden ser el resultado de su transformación, $\chi^2 = (error/sd)^2$. Por la teoría, sabemos que la desviación típica se estable cuando tenemos suficientes datos y, por tanto, los valores extremos de χ^2 son causa de un error no ajustado.
3. El mismo comportamiento también se puede ver en `qqplot`. Vemos que los cuantiles muestrales no es una recta, sino una forma de S plana.
4. Finalmente, el estimador de DW está muy cerca al valor 2 con un p valor apriximadamente 0. En este caso, presenta una pequeña autocorrelación positiva en los errores. También se puede observar en las gráficas.

Basando estas salidas, el modelo planteado no ha sido validado. Supongamos que lo fuera, para extrar la estimación de la conductividad se necesita el método `extractMeasures`. Este método nos permite tener las medias en una variable y, también, en la representación gráfica junto con el offset estimado.

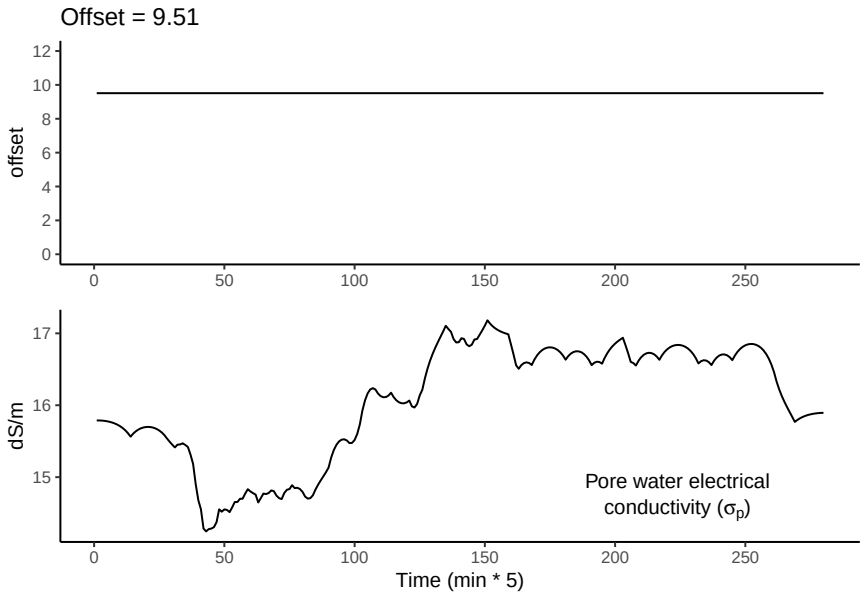


Figura 3.5: La conductividad eléctrica mediante DLM

Model de Hilhorst y comparación con DLM De una manera similar, podemos obtener el modelo de Hilhorst de las señales brutas. Dando el offset anterior 9.51, la estimación de la conductividad eléctrica de agua para el modelo de Hilhorst es:

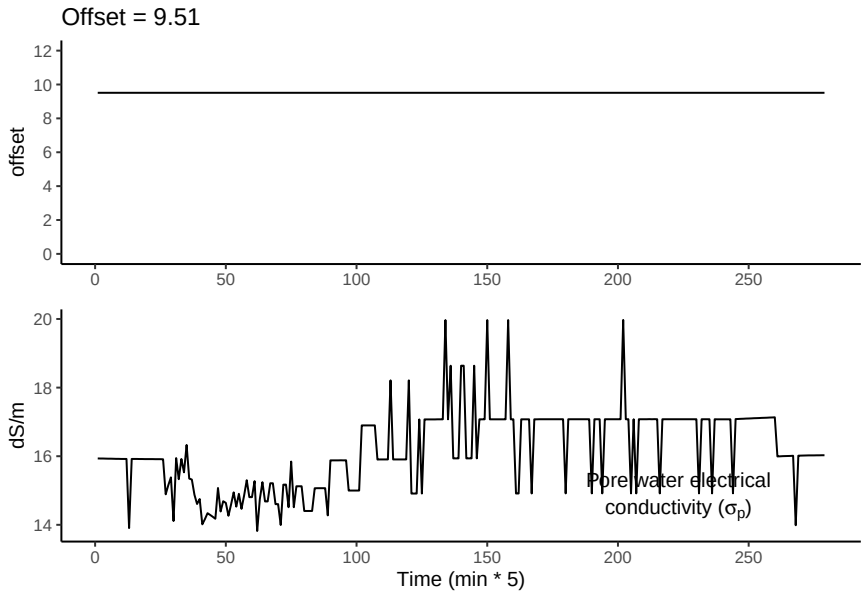


Figura 3.6: La conductividad eléctrica mediante Hilhorst

La estimación instáneo del modelos Hilhorst es menos suave que DLM y, finalmente, podemos sobreponer estas dos gráficas para hacer una comparación, ver figura 3.7. Tal vez, la representación en línea en ambos casos tiene un compotamiento similar, pero DLM tiene una estiamción más suave que el otro.

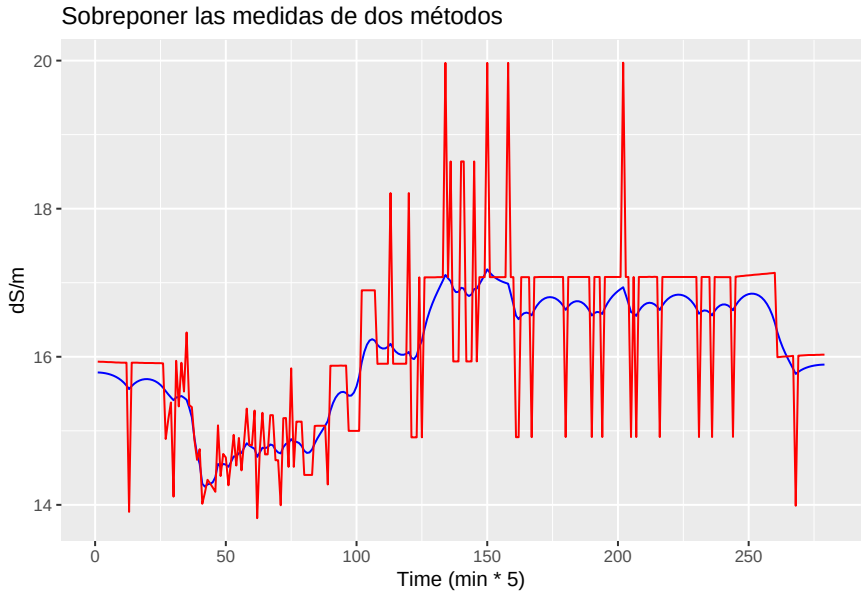


Figura 3.7: Gráfica conjunta de dos medidas

Simulación mediante RNN En el final de esta sección, dejamos la instrucción para general el modelo de RNN. La simulación mediante RNN es un prototipo y aún no está completo, pero funciona.

```
## build RNN
compile_options <- list()
hyperparameters <- list(batch_size = c(16, 32, 64),
                        epochs = c(10, 20, 50))
preprocess_method <- "maxmin"
lag <- 12
model_path <- "~/Desktop/pans_model"
myRNN <- buildRNN(object = myclass, compile_options = list(),
                  hyperparameters = hyperparameters,
                  preprocess_method = preprocess_method,
                  lag = lag, model_path = model_path)

RNN_predict(myRNN, sigb = new_sigb, epsp = new_epsp)
```

3.3. Resumen y mejoras

Basando en las salidas, la primera versión `sm4sd` tiene su insuficiencia y aún require una mejora en el futuro. A pesar de su insufuciencia, este paquete da un paso adelante sobre el análisis de la conductividad eléctica del agua en el terreno cultivo. El método dlm nos deja un análisis más sofisticado en tiempo real y, además, la definición en clase `S4` facilitar el uso del paquete a aplicar las investigaciones futuras.

Con el fin de completar este paquete, los siguientes puntos se tienen de considerar:

1. la autodetección y el tractamiento de los atípicos no observados en los datos sensóricos;
2. la incorporación de un extra argumento para controrar el tipo de offset, variante o constante;
3. la incorporación del proceso de paralelización en la búsqueda automática para aumentar el rendimiento;
4. la posibilidad de definir una ventana distinta de restraso para cada indicador individual;
5. y, finalmente, definir el optimizador EKF (el filtro de Kalman extendido) a las RNN.

Los puntos mencionados no han sido implementados en esta versión ya que se plantean como una extensión de la implementación de este trabajo.. Entre estos puntos, la implementación de EKF a RNN será un gran obstaculo por dos motivos. El primero, la definición de EFK deberá llevar a cabo en el entorno de *Python* porque lo es el núcleo de *Keras*. El segundo, la incorporación de optimizador a los modelos keras también es problemática.

Conclusion

El objetivo final de esta tesis es crear un paquete en R sobre el análisis de los datos de sensória mediante los modelos dinámicos. En este trabajo, hemos hecho un primer paso con el ejemplo del análisis de la conductividad eléctica de agua en el terreno cultivo. El paquete `sm4sd` ha mostrado su poder y opera correctamente. Especialmente, este paquete facilita el uso de la aplicación de dlm al caso real y, incluso, introduce formalidad y mejoras de los códigos en el artícuo Basem et al. (2018). Vinculando a este paquete, dos temas complejos han sido tractados que son el estudio de los modelos dinámicos lineales y la aplicación de las redes neuronales recurrentes.

Tras de realizar esta memoria, he consolidado los conocimientos de DLM y el método de Kalman Filter debido a una investigación profunda de ellos. Adicionalmente, los puntos estuadios pueden servir como el complementario de de la segunda parte de la asignatura *Time Series*. En decadas recientes, DLM y KF han sido consideradamente importantes en diferentes ámbitos y, incluso, el uso del KF para caso no lineal se extiende a las redes neuronales. En concreto, el estudio de RNN entrenado por el filtro de Kalman extendido se considera en esta tesis, pero no llegamos más allá de su estructura y la configuración para no extender un trabajo sin límite.

A parte de estos modelos, este trabajo hace una pequeña introducción sobre los conceptos básicos sobre R como los diferentes sitemas orientados a objetos, el uso de la función genérica y la construcción de un paquete. Estos elementos forman el pilar del paquete `sm4sd`. Final de todo, unas explicaciones detalladas sobre los objetos definidos en el paquete y una aplicación real.

En conclusión, la construcción del paquete en R es la idea central de esta tesis. Sin embargo, existe una limitación en este verisón porque solo sirve para el análisis del conductiviad eléctrica del agua. Con el fin de generalizar este resulto a todos los posibles análisis, una reconstrucción del paquete basando en una entrada no fija y una definición dinámicos de las variables latentes y la respuesta mediante las formulas. Por su complejidad, esta reconstrucción del paquete `sm4sd` pueda dar lugar una tesis posterior..

Referencias

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/

Aljoumani, B., Sanchez-Espigares, J. A., & Wessolek, G. (2018). Estimating pore water electrical conductivity of sandy soil from time domain reflectometry records using a time-varying dynamic linear model. *Sensors*, *18*(12), 4403.

Campagnoli, P., Petris, G., & Petrone, S. (2009). *Dynamic linear models with r*. Springer.

Chollet, F., & others. (2015). Keras. https://github.com/fchollet/keras; GitHub.

Genolini, C. (2008). *A (not so) short introduction to s4*. Technical report, The R-Project for Statistical Computing.

Harvey, A. C. (1990). *Forecasting, structural time series models and the kalman filter*. Cambridge University Press.

Hilhorst, M. A. (2000). A pore water conductivity sensor. *Soil Science Society of America Journal*, *64*(6), 1922–1925.

Huber, W., Carey, V. J., Gentleman, R., Anders, S., Carlson, M., Carvalho, B. S., ... Morgan, M. (2015). Orchestrating high-throughput genomic analysis with Bioconductor. *Nature Methods*, *12*(2), 115–121. http://www.nature.com/nmeth/journal/v12/n2/full/nmeth.3252.html

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems.

Lipton, Z. C., Berkowitz, J., & Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. *arXiv Preprint arXiv:1506.00019*.

Oancea, B., Andrei, T., Dragoescu, R. M., & others. (2016). An r implementation of a recurrent neural network trained by extended kalman filter. *Romanian Statistical Review*, *64*(2), 125–133.

Ocaña-Riola, R. (2017). La necesidad de convertir la estadística en profesión regulada. *Estaddística Española*, *59*(194), 193–212.

Oliveira, M. A. de. (2012). An application of neural networks trained with kalman filter variants (ekf and ukf) to heteroscedastic time series forecasting. *Applied Mathematical Sciences*, *6*(74), 3675–3686.

Persson, M. (2002). Evaluating the linear dielectric constant-electrical conductivity model using time-domain reflectometry. *Hydrological Sciences Journal*, *47*, 269–277. http://doi.org/10.1080/02626660209492929

Petris, G. (2009). Dlm: An r package for bayesian analysis of dynamic linear models. *University of Arkansas*.

Petris, G., & An, R. (2010). An r package for dynamic linear models. *Journal of Statistical Software*, *36*(12).

Petris, G., & Petrone, S. (2011). State space models in r. *Journal of Statistical Software*, *41*(4).

R Core Team. (2017). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/

Reid, I. (2012). Estimation ii. http://www.robots.ox.ac.uk/~ian/Teaching/Estimation/LectureNotes2.pdf

Singhal, S., & Wu, L. (1989). Training multilayer perceptrons with the extended kalman algorithm. In *Advances in neural information processing systems* (pp. 133–140).

Stroustrup, B. (2012). Software development for infrastructure. *IEEE Computer*, *45*(1), 47–58.

Tusell, F., & others. (2011). Kalman filtering in r. *Journal of Statistical Software*, *39*(2), 1–27.

West, M., & Harrison, J. (2006). *Bayesian forecasting and dynamic models*. Springer Science & Business Media.

Wickham, H. (2015). *R packages: Organize, test, document, and share your code*. .º'Reilly Media, Inc.".

Wickham, H. (2019). *Advanced r*. CRC press.

Wickham, H., Danenberg, P., & Eugster, M. (2018). *Roxygen2: In-line documentation for r*. https://CRAN.R-project.org/package=roxygen2

Apéndice A: Métodos

En este repositorio, https://github.com/4301350/sm4sd, se encuentra el directorio original del proyecto de la libreria sm4sd incluyendo la documentación en roxygen2. En el apéndice A solo está disponible la definición de los métodos exepto [, print y show.

buildClass

```
# Función genérica
setGeneric("buildClass",
           function(object, method, freq, ind, lagMax, verify, parallel)
             standardGeneric("buildClass"))
```

```
# método y su definición interna para `RS`
setMethod(f = "buildClass",
          signature = c("RS", "character", "ANY", "ANY", "ANY", "ANY", "ANY"),
          definition = .RS.buildClass)

.RS.buildClass <- function(object, method, freq, ind,
                           lagMax, verify, parallel){
  call <- match.call()

  method <- tolower(method)
  # method forecast is not available
  if( !(method %in% c("forecast", "dlm", "hl")) ) {
    stop("No method has define por this index.")
  }
  if( method == "dlm" ){
    index <- ind
    if( !is.numeric(freq) |  freq %% 1 != 0 | freq < 0 ){
      stop("Frequency must a integer value bigger o equal to 0.")
    }
    if( !is.list(index) ){
      stop("The event should be a list.")
    }
```

```
    if(!is.numeric(lagMax) |  length(lagMax) != 1){
      stop("lagMax must be length one numeric variable.")
    }
    if(!is.logical(verify) | !is.logical(parallel)){
      stop("verify & parallel must be logical.")
    }

    mod <- .RS.buildClass.buildDLM(call = call, object = object,
                                   freq = freq, index = index,
                                   lagMax = lagMax, verify = verify,
                                   parallel = parallel)
  }
  if( method == "hl" ){
    offset <- ind
    if( length(offset) != 1 | offset < 0 ){
      stop("The offset should be no negative length one numeric variable.")
    }

    mod <- .RS.buildClass.buildMP(call = call, object = object, offset = offset)
  }
  mod
}
# build SM.dlm
.RS.buildClass.buildDLM <- function(object, freq, index,
                                    lagMax, verify, parallel, call){
  class.dlm <- SM.dlm(call = call,
                      freq = freq,
                      indicators = index,
                      lagMax = lagMax,
                      verify = verify,
                      parallel = parallel,
                      object)
  class.dlm
}
# build SM.HL
.RS.buildClass.buildHL <- function(object, offset, call){
  class.HL <- SM.HL(call = call,
                    offset = offset ,
                    object)
  class.HL
}
```

buildRNN

```
# Función genérica
setGeneric("buildRNN",
           function(object, compile_options, hyperparameters,
                    preprocess_method, lag, model_path)
             standardGeneric("buildRNN"))
```

```
# método y su definición interna para `RS`
setMethod(f = "buildRNN",
          signature = c("RS", "list", "list",
                        "character", "numeric", "character"),
          definition = .SM.buildRNN)

.SM.buildRNN <- function(object, compile_options, hyperparameters,
                         preprocess_method = c("normalized", "maxmin"),
                         lag,
                         model_path){
  call <- match.call()
  # redefine compile_options and hyperparameters
  # including default value for missing info
  res <-.SM.buildRNN.parameterDefine(compile_options, hyperparameters)
  compile_aux <- res$compile_options
  hyper_aux <- res$hyperparameters

  # Define the preprocess parameters
  if( ! preprocess_method %in% c("maxmin", "normalized") ){
    stop("Method introduced has not defined for this function.")
  }
  res <- .SM.buildRNN.preprocessParam(object, x1_name = "sigb",
                                      x2_name = "epsp", y_name = "epsb",
                                      method = preprocess_method,
                                      lag = lag)
  X <- res$data$X_data
  Y <- res$data$Y_data
  preprocess_parameter <- res$preprocess_parameter

  param.grid <- expand.grid(batch_size = hyper_aux$batch_size,
                            epochs = hyper_aux$epochs,
                            test_split = hyper_aux$test_split,
                            validation_split = hyper_aux$validation_split,
                            metric = NA)
  if( compile_aux$metrics  == "mean_absolute_percentage_error"){
    names(param.grid)[5] <- "mape"
  }
```

```
for( i in seq(nrow(param.grid)) ){

  split_data <- .SM.buildRNN.split(X, Y, param.grid$test_split[i],
                                   param.grid$validation_split[i])

  model <- keras_model_sequential()
  model %>%
    layer_dense(input_shape = dim(X)[2:3], units = dim(X)[2])
  model %>%
    layer_simple_rnn(units = dim(X)[2])
  model %>%
    layer_dense(units = 1, activation = "linear")

  model %>% compile(
    optimizer = compile_aux$optimizer,
    loss = compile_aux$loss,
    metrics = compile_aux$metrics
  )

  trained_model <- model %>% keras::fit(
    x = split_data$X_train,
    y = split_data$Y_train,
    batch_size = param.grid$batch_size[i],
    epochs = param.grid$epochs[i],
    validation_data = list(split_data$X_test, split_data$Y_test),
    shuffle = F
  )
  ypred <- model %>%
    predict(split_data$X_valid)

  param.grid[i, 5] <- sum(abs(ypred - split_data$Y_valid))

  model %>%
    save_model_hdf5(paste0("model_", i))
}

min.index <- which.min(param.grid[, 5])
model_name <- paste0("model_", min.index)
new_model <- load_model_hdf5(model_name)

files <- list.files()
files <- files[grep("model_", files)]
file.remove(files)
```

```
  new_model %>%
    save_model_hdf5(paste0(model_path))

  tracking <- param.grid[, c(1, 2, 5)]
  tracking <- tracking[order(tracking$mape), ]

  SM.rnn(call = call, model_path = model_path,
         compile_options = compile_aux,
         optim_hyperparameters = list(
           batch_size = param.grid$batch_size[min.index],
           epochs = param.grid$epochs[min.index]),
         preprocess_parameter = res$preprocess_parameter,
         tracking = tracking, object)
}

.SM.buildRNN.parameterDefine <- function(compile_options, hyperparameters){
  # define default compile_options for missing data:
  default_compile <- c("optimizer", "loss", "metrics")
  default_compile_value <- c("adam", "mean_absolute_percentage_error",
                             "mean_absolute_percentage_error")
  compile_names <- names(compile_options)

  optimizer_aux <- list()
  for(i in seq(length(default_compile))){
    if(!default_compile[i] %in% compile_names){
      optimizer_aux[[i]] <- default_compile_value[i]
    }else{
      n_index <- which(compile_names == default_compile[i])
      optimizer_aux[[i]] <- compile_options[[n_index]]
    }
  }
  names(optimizer_aux) <- default_compile

  # Define default hyperparameters for missing data:
  default_hyper <- c("batch_size", "epochs", "test_split", "validation_split")
  default_value <- c(32, 10, 0.3, 0.3)
  hyper_names <- names(hyperparameters)

  hyper_aux <- list()
  for(i in seq(length(default_hyper))){
    if(!default_hyper[i] %in% hyper_names){
      hyper_aux[[i]] <- default_value[i]
    }else{
      n_index <- which(hyper_names == default_hyper[i])
```

```
      hyper_aux[[i]] <- hyperparameters[[n_index]]
    }
  }
  names(hyper_aux) <- default_hyper

  res <- list(compile_options = optimizer_aux, hyperparameters = hyper_aux)
  res
}

.SM.buildRNN.preprocessParam <- function(object, x1_name = "sigb",
                                         x2_name = "epsp",
                                         y_name = "epsb", method,
                                         lag = 0){
  X <- as.matrix(cbind(object[x1_name], object[x2_name]))
  Y <- as.matrix(object[y_name])

  if( method == "normalized" ){
    pre_param_x <- list(center = attr(scale(X), "scaled:center"),
                        scale = attr(scale(X), "scaled:scale"),
                        var_name = c(x1_name, x2_name))
    pre_param_y <- list(center = attr(scale(Y), "scaled:center"),
                        scale = attr(scale(Y), "scaled:scale"),
                        var_name = c(y_name))
  }else if( method == "maxmin"){
    pre_param_x <- list(center = attr(scale(X), "scaled:center"),
                        scale = attr(scale(X), "scaled:scale"),
                        max = apply(scale(X), 2, max) * 1.1 ,
                        min = apply(scale(X), 2, min) * 1.1,
                        var_name = c(x1_name, x2_name))
    pre_param_y <- list(center = attr(scale(Y), "scaled:center"),
                        scale = attr(scale(Y), "scaled:scale"),
                        max = apply(scale(Y), 2, max) * 1.1,
                        min = apply(scale(Y), 2, min) * 1.1,
                        var_name = c(y_name))
  }else{
    stop("The method is not valid.")
  }

  res <- .SM.buildRNN.preprocess(X = X, Y = Y, method = method,
                                 pre_param_x = pre_param_x,
                                 pre_param_y = pre_param_y,
                                 lag = lag)

  list(data = res,
```

```
      preprocess_parameter = list(X = pre_param_x, Y = pre_param_y,
                                  method = method, lag = lag))
}

.SM.buildRNN.preprocess <- function(X, Y = NULL, method,
                                    pre_param_x, pre_param_y, lag){

  if(method == "normalized"){
    res <- .SM.buildRNN.normMethod(X, Y, pre_param_x, pre_param_y)
  }
  if(method == "maxmin"){
    res <- .SM.buildRNN.normMethod(X, Y, pre_param_x, pre_param_y)
    res <- .SM.buildRNN.maxminMethod(res$X, res$Y, pre_param_x, pre_param_y)
  }

  if( lag != 0 ){
    aux_x <- res$X

    x1 <- matrix(aux_x[1 : (1 + lag), 1], ncol = lag + 1)
    x2 <- matrix(aux_x[1 : (1 + lag), 2], ncol = lag + 1)
    for(i in seq(2, nrow(aux_x) - lag)){
      x1 <- rbind(x1, aux_x[i : (i + lag), 1])
      x2 <- rbind(x2, aux_x[i : (i + lag), 2])
    }
    aux_x <- cbind(x1, x2)
    dim(aux_x) <- c(dim(aux_x), 1)

    if( !is.null(Y)){
      aux_y <- res$Y
      aux_y <- matrix(aux_y[(1 + lag) : nrow(aux_y)])
    }
  }

  if( !is.null(Y) ){
    list(X_data = aux_x, Y_data = aux_y)
  }else{
    list(X_data = aux_x)
  }
}

.SM.buildRNN.normMethod <- function(X, Y, pre_param_x, pre_param_y){

  X <- sweep(X - rep(pre_param_x$center, each = nrow(X)), 2,
             pre_param_x$scale, FUN = "/")
```

```
  if( !is.null(Y) ){
    Y <- sweep(Y - rep(pre_param_y$center, each = nrow(Y)), 2,
               pre_param_y$scale, FUN = "/")
  }
  list(X = X, Y = Y)
}


.SM.buildRNN.maxminMethod <- function(X, Y, pre_param_x, pre_param_y){

  X <- sweep(X - rep(pre_param_x$min, each = nrow(X)), 2,
             pre_param_x$max - pre_param_x$min, FUN = "/")
  if( !is.null(Y) ){
    Y <- sweep(Y - rep(pre_param_y$min, each = nrow(Y)), 2,
               pre_param_y$max - pre_param_y$min, FUN = "/")
  }
  list(X = X, Y = Y)
}

.SM.buildRNN.split <- function(X, Y, split_rate, validation_rate){
  if(length(dim(X)) == 3){
    X <- X[, , 1]
  }

  if( is.matrix(X) ){
    n <- nrow(X)
    train_index <- seq(n * (1 - validation_rate))
    train_train <- seq(max(train_index) * (1 - split_rate))
    train_test <- train_index[-train_train]
  }

  X_train <- X[train_train, ]
  dim(X_train) <- c(dim(X_train), 1)

  X_test <- X[train_test, ]
  dim(X_test) <- c(dim(X_test), 1)

  X_validation <- X[-c(train_index), ]
  dim(X_validation) <- c(dim(X_validation), 1)

  list(X_train = X_train, X_test = X_test,
       X_valid = X_validation,
       Y_train = Y[train_train], Y_test = Y[train_test],
       Y_valid = Y[-train_index])
```

```
}
```

extractMeasures

```
# Función genérica
setGeneric("extractMeasures",
           function(object, plot) standardGeneric("extractMeasures"))
```

```
# método y su definición interna para `SM.dlm`
setMethod(f = "extractMeasures",
          signature = c("SM.dlm", "logical"),
          definition = .SM.dlm.extractMeasures)

.SM.dlm.extractMeasures <- function(object, plot){
  cat("In order to extract measures, the model must to be fitted.\n")
  stop("Fit the model first.")
}
```

```
# método y su definición interna para `SM.dlm.fitted`
setMethod(f = "extractMeasures",
          signature = c("SM.dlm.fitted", "logical"),
          definition = .SM.dlm.fitted.extractMeasures)

.SM.dlm.fitted.extractMeasures <- function(object, plot){
  smoothed <- object@smoothed$s

  measures <- data.frame(offset = round(smoothed[, 1], 2),
                         conductivity = 1 / smoothed[, 2])

  plots <- .extractMeasures.plot(measures)
  p1 <- plots[[1]]
  p2 <- plots[[2]]

  if( plot ){
    options(warn = -1)
    grid.arrange(p1, p2, ncol = 1)
    options(warn = 0)
  }

  n <- nrow(measures)
  if( measures$offset[n] == mean(measures[round(n/2):n,]$offset) ){
    output <- list(mean(measures$offset), measures$conductivity)
  }else{
```

```
    output <- list(measures$offset, measures$conductivity)
  }
  names(output) <- names(measures)
  output
}
```

```
# método y su definición interna para `SM.HL`
setMethod(f = "extractMeasures",
          signature = c("SM.HL", "logical"),
          definition = .SM.HL.extractMeasures)

.SM.HL.extractMeasures <- function(object, plot){
  offset <- object@offset
  yt <- object@epsb
  xt <- object@epsp * object@sigb

  conductivity <- xt / (yt - offset)

  measures <- data.frame(offset = rep(offset, length(yt)),
                         conductivity = conductivity)

  plots <- .extractMeasures.plot(measures)

  p1 <- plots[[1]]
  p2 <- plots[[2]]
  if( plot ){
    options(warn = -1)
    grid.arrange(p1, p2, ncol = 1)
    options(warn = 0)
  }


  output <- list(mean(measures$offset), measures$conductivity)

  names(output) <- names(measures)
  output
}
```

```
# Función interna común
.extractMeasures.plot <- function(df){
  measures <- df
  p1 <- ggplot(measures, aes(x = seq(length(offset)), y = offset)) +
    geom_line() +
    labs(y = "offset",
```

```
        title = paste0("Offset = ",
                       format(round(measures$offset[1], 2), nsmall = 2))) +
    coord_cartesian(ylim = c(0, 12)) +
    scale_x_continuous(breaks = seq(0, nrow(measures), 50)) +
    scale_y_continuous(breaks = seq(0, 12, 2)) +
    theme(axis.title.x = element_blank(),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.background = element_blank(),
          axis.line = element_line(colour = "black"))

  conduc <- measures$conductivity
  conduc <- conduc[is.finite(conduc)]
  dif <- (max(conduc) - min(conduc)) / 10
  y_aux <- min(conduc) + dif
  p2 <- ggplot(measures,
               aes(x = seq(length(conductivity)), y = conductivity)) +
    geom_line() +
    labs(x = "Time (min * 5)",
         y = "dS/m") +
    scale_x_continuous(breaks = seq(0, nrow(measures) ,50)) +
    annotate("text", x = nrow(measures) * 0.8, y = y_aux,
             label = expression("Pore water electrical\n",
                                paste("conductivity (", sigma[p], ")"))) +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.background = element_blank(),
          axis.line = element_line(colour = "black"))
  list(p1, p2)
}
```

fit

```
# Genérica
setGeneric("fit", function(object) standardGeneric("fit"))
```

```
# método y su definición interna para `SM.dlm`
setMethod(f = "fit", signature = c("SM.dlm"), definition = .SM.dlm.fit)

.SM.dlm.fit <- function(object){
  i_call <- match.call()
  class.fitted <- .SM.dlm.train(object, i_call)
  class.fitted
```

```
}

.SM.dlm.train <- function(object, i_call){
  epsb <- object@epsb
  epsp <- object@epsp
  sigb <- object@sigb
  freq <- object@freq
  index <- object@indicators
  lagMax <- object@lagMax

  # List length
  n <- length(index)
  # Total number of list elements
  n_total <- length(unlist(index))
  # Registre the length of each element of index
  n_index <- c()
  if(length(index) != 0 ){
    for(i in seq(length(index))){
      n_index <- c(n_index, rep(i, length(index[[i]])))
    }
  }
  # Define xt and yt
  yt <- epsb
  xt <- epsp * sigb

  if(!is.null(n_index)){
    # Define the grid for tunning parameter
    param_grid <- expand.grid(replicate(n_total, seq(lagMax), simplify = F))
    colnames(param_grid) <- names(unlist(index))

    # Define a dataframe storing parameter and MLE value.
    df <- data.frame()

    cat("Fitting")
    for(i in seq(nrow(param_grid))){
      lag_list <- split(as.numeric(param_grid[i,]), n_index)
      names(lag_list) <- names(index)

      build <- function(x){
        dlm <- .SM.dlm.buildFunction(x = x, xt = xt, yt = yt, freq = freq,
                                     index = index, lag_list = lag_list)
        dlm
      }
```

```
    # build dlm model
    param_aux <- rep(-1, length(index))
    tryCatch(
      {
        fit <- dlmMLE(yt, parm = c(-1, -1, param_aux), build = build)
        param <- c(as.integer(param_grid[i,]), fit$value, fit$par)
        names(param) <- c(names(param_grid), "LogLik", "V", "W", names(index))

        df <- rbind(df, as.data.frame(t(param)))
      }, error = function(e){
      }
    )
    cat(".")
  }
}else{
  df <- data.frame()

  build <- function(x){
    dlm <- .SM.dlm.buildFunction(x = x, xt = xt, yt = yt, freq = freq,
                                 index = index, lag_list = lag_list)
    dlm
  }
  fit <- dlmMLE(yt, parm = c(-1, -1), build = build)
  param <- c(fit$value, fit$par)
  names(param) <- c(names(param_grid), "LogLik", "V", "W")

  df <- rbind(df, as.data.frame(t(param)))
  cat(".")
}
cat("\nDone.\n")
min_logLik <- df[which.min(df$LogLik),]
LogLik_pos <- which(names(min_logLik) == "LogLik")
n_max <- ncol(min_logLik)
param <- unlist(min_logLik[, (LogLik_pos + 1) : n_max])
# names(param) <- colnames(min_logLik)[(LogLik_pos + 1) : n_max]
if(length(index) != 0){
  lag_list <- split(unname(unlist(min_logLik[, 1: LogLik_pos - 1])), n_index)
  names(lag_list) <- names(index)
}else{
  lag_list <- list()
}
dlmFit <- build(param)

# extrac the parameter and their names
```

```
param <- as.list(param)

smoothed <- dlmSmooth(yt, dlmFit)
filtered <- dlmFilter(yt, dlmFit)

# Order the tracking list according the -LogLikelihood
df <- df[order(df$LogLik), ]

# Stationality test
if( freq >= 2 ){
  cat("Stationality Test is applied: ")
  build_2 <- function(x){
    dlm <- .SM.dlm.buildFunction(x = x, xt = xt, yt = yt, freq = 1,
                                 index = index, lag_list = lag_list)
    dlm
  }
  fit_2 <- dlmMLE(yt, parm = c(-1, -1, param_aux), build = build_2)

  # Compute the -loglikelihood for two models
  log_1 <- df[1, ]$LogLik
  log_2 <- fit_2$value

  # Statistic of Wald test, it is assymptotically a chisq
  # with 0 degree of freedoom.
  LR <- 2 * (log_2 - log_1)
  chi.value <- pchisq(LR, 0, lower.tail = F)

  if(chi.value < 0.05){
    stationalitySign <- "TRUE"
    cat("The seasonality is significant.\n")
  }else{
    stationalitySign <- "FALSE"
    cat("The seasonality is not significant.\n")
  }
}else{
  stationalitySign <- "NULL"
}

class(filtered) <- "list"
class.fitted <- SM.dlm.fitted(call = i_call,
                              parameters = param,
                              filtered = filtered,
                              smoothed = smoothed,
                              lags = lag_list,
```

```
                                    seasonalSign = stationalitySign,
                                    tracking = df,
                                    object)

  class.fitted
}

.SM.dlm.buildFunction <- function(x, xt, yt, freq, index, lag_list) {
  if(freq == 0 | freq == 1){
    myMod <- dlmModReg(xt)
  }else{
    myMod <- dlmModReg(xt) + dlmModSeas(freq, dV = 0)
  }
  myMod$JFF[, 2] <- matrix(3)
  myMod$JV <- matrix(4)
  myMod$JW <- diag(c(1, 2))
  X1 <- matrix(rep(c(0, exp(x[2])), length(yt)), ncol = 2, byrow = T)
  X2 <- matrix(rep(exp(x[1]), length(yt)))
  myMod$X <- cbind(X1, myMod$X, X2)
  if(length(index) != 0){
    for(i in seq(length(index))){
      for(j in seq(length(index[[i]]))){
        j_init <- index[[i]][j]
        if(j + lag_list[[i]][j] - 1 > nrow(myMod$X)){
          j_final <- nrow(myMod$X)
        }else{
          j_final <- j_init + lag_list[[i]][j] - 1
        }
        myMod$X[j_init : j_final, 4] <- myMod$X[j_init : j_final, 4] +
          X2[j_init : j_final, 1] * exp(x[2 + i])
      }
    }
  }
  myMod
}
```

getMod

```
# Genérica
setGeneric("getMod", function(object, name) standardGeneric("getMod"))
```

```
# método y su definición interna para `SM.dlm.fitted`
setMethod(f = "getMod", signature = c("SM.dlm.fitted", "missing"),
          definition = .SM.dlm.fitted.getdlm)
```

```
.SM.dlm.fitted.getdlm <- function(object){
  epsb <- object@epsb
  epsp <- object@epsp
  sigb <- object@sigb
  freq <- object@freq
  index <- object@indicators

  # Define xt and yt
  yt <- epsb
  xt <- epsp * sigb

  param <- unlist(object@parameters)
  lag_list <- object@lags
  build <- function(x){
    dlm <- .SM.dlm.buildFunction(x, xt, yt, freq, index, lag_list )
    dlm
  }
  dlmFit <- build(param)

  dlmFit
}
```

plot

```
# método y su definición interna para `RS`
setMethod(f = "plot",
          signature = c("RS", "missing"),
          definition = .RS.plot)

## Internal definition of plot methods

.RS.plot <- function(x, y, ...){
  sigb <- x@sigb
  epsb <- x@epsb
  epsp <- x@epsp

  series <- data.frame(sigb = as.vector(sigb),
                       epsb = as.vector(epsb),
                       epsp = as.vector(epsp))
  timeSeries::plot(ts(series),
                   plot.type = "multiple",
                   main = "plot of 'sigb', 'epsb' and 'epsp'")
  invisible()
```

```
}
```

plotCor

```
# Genérica
setGeneric("plotCor", function(x, groups, ...) standardGeneric("plotCor"))
```

```
# método y su definición interna para `SM.HL`
setMethod(f = "plotCor",
          signature = c("RS", "numeric"),
          definition = .RS.plot.corr)

.RS.plot.corr <- function(x, groups, ...){
  group <- NULL

  yt <- x@epsb
  xt <- x@epsp * x@sigb

  if(groups %% 1 != 0 | groups <= 0){
    stop("Value of groups must to be a posive integer.")
  }
  if( groups > 6){
    groups <- 6
    warning("Group is exceeded 6, set to 6!")
  }

  df <- data.frame(xt = xt, yt= yt)
  if(groups == 1){
    p  <- ggplot(data = df, mapping = aes(x = yt, y = xt))
    p <-  p + geom_point(color = "blue") +
      labs(title = "epsilon-b vs. epsilon-p * sigma-b",
           x = "Epsilon-b",
           y = "Epsilon-p * Sigma-b") +
      theme(panel.grid.major = element_blank(),
            panel.grid.minor = element_blank(),
            panel.background = element_blank(),
            axis.line = element_line(colour = "black"))

    p
  }else{
    n <- length(yt)
    k <- ceiling(n / groups)
    df$group <- as.factor(rep(1 : groups, each = k)[1 : n])
```

```
    p  <- ggplot(data = df,
                 mapping = aes(x = yt, y = xt, color = group, shape = group))
    p <- p + geom_point() +
      labs(title = "epsilon-b vs. epsilon-p * sigma-b",
           x = "Epsilon-b",
           y = "Epsilon-p * Sigma-b")

    p <- p + geom_smooth(method = "lm", se = F) +
      theme_bw() +
      expand_limits( y = 0 ) +
      theme(panel.grid.major = element_blank(),
            panel.grid.minor = element_blank(),
            panel.background = element_blank(),
            axis.line = element_line(colour = "black"))
    p
  }
}
```

residualDiag

```
# Genérica
setGeneric("residualDiag",
           function(object,  gamma)
             standardGeneric("residualDiag"))
```

```
# método y su definición interna para `SM.dlm.fitted`
setMethod(f = "residualDiag",
          signature = c("SM.dlm.fitted", "numeric"),
          definition = .SM.dlm.fitted.residualDiag)

.SM.dlm.fitted.residualDiag <- function(object, gamma){

  res <- .SM.dlm.fitted.getResid(object, type = "raw", sd = TRUE)
  if(gamma > 1 & gamma < 0){
    stop("The significance level between 0 and 1.")
  }

  df.res <- data.frame(ind = seq(length(res$res)),
                       resid = res$res,
                       sd = res$sd,
                       norm_square = (res$res / res$sd)**2)

  p1 <- .SM.dlm.fitted.confBand(df.res, gamma)
```

```
  p2 <- .SM.dlm.fiited.chis2(df.res)
  p3 <- .SM.dlm.fitted.QQPlot(df.res)

  options(warn = -1)
  grid.arrange(p1, p2, p3, ncol = 2)
  options(warn = 0)

  .SM.dlm.fitted.tests(df.res$resid)
  invisible()
}

# get error
.SM.dlm.fitted.getResid <- function(object,
                                    type = c("standardized", "raw"),
                                    sd = TRUE){
  filtered <- object@filtered
  class(filtered) <- "dlmFiltered"

  if(type != "standardized" & type != "raw"){
    stop("No recognised type of error!")
  }else{
    resid <- residuals(filtered, type = type, sd = sd)
  }

  if( sd ){
    resid$res <- resid$res[-1]
    resid$sd <- resid$sd[-1]
  }else{
    resid <- resid[-1]
  }

  resid
}
# Confidence band
.SM.dlm.fitted.confBand <- function(df.res, gamma){
  ind <- resid <- sd <- NULL

  coef <- qnorm((1-gamma)/2 + gamma)
  ratio <- round(sum(df.res$resid >= -1 * coef * df.res$sd &
                       df.res$resid <= coef * df.res$sd)/nrow(df.res), 2)

  p1 <- ggplot(data = df.res, aes(x = ind, y = resid)) +
    geom_point() +
    geom_line(aes(y = sd * coef), col = "blue") +
```

```
    geom_line(aes(y = - 1 * sd * coef), col = "blue") +
    coord_cartesian(ylim = 1.5 * c(min(df.res$resid), max(df.res$resid))) +
    labs(title = paste0("Confidence Band Plot: ", gamma),
         ylabel = "Resid",
         caption = paste0(ratio * 100, "% inside of the band"), col = "red") +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.background = element_blank(),
          axis.line = element_line(colour = "black"))

  p1
}

# chisq2 test
.SM.dlm.fiited.chis2 <- function(df.res){
  ind <- norm_square <- ma_chi <- NULL

  ma <- function(x){
    aux <- x[1]
    for (i in seq(2, length(x))){
      aux <- c(aux, mean(x[1:i]))
    }
    aux
  }
  df.res$ma_chi <- ma(df.res$norm_square)

  p2 <- ggplot(data = df.res, aes(x = ind, y = norm_square)) +
    geom_point() +
    geom_line(aes(y = ma_chi), col = "red") +
    geom_line(aes(y = 1), col = "blue") +
    labs(title = "Chisq Test Plot",
         ylab = "Norm Error",
         caption = paste0("Mean of chisq estimator: ",
                          round(mean(df.res$norm_square)))) +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.background = element_blank(),
          axis.line = element_line(colour = "black")) +
    annotate("text", 1, 1, vjust = -1, label = "y = 1")

  p2
}

## qqplot
```

```
.SM.dlm.fitted.QQPlot <- function(df.res){
  resid <- sd <- NULL

  p3 <- ggplot(df.res, aes(sample = resid / sd)) +
    stat_qq() +
    stat_qq_line() +
    labs(title = "Norm Q-Q plot",
         x = "Theoretical Quantiles",
         y = "Sample Quantiles") +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.background = element_blank(),
          axis.line = element_line(colour = "black"))

  p3
}

## autocorrelation test
.SM.dlm.fitted.tests <- function(resid){
  y <- resid[2:length(resid)]
  x <- resid[1:(length(resid) -1 )]
  dw_test <- dwtest(y ~ x)

  cat(paste0("    Durbin-Watson Test : ",  dw_test$data.name, "\n\n"))
  cat(paste0(names(dw_test$statistic), " = ",
             round(dw_test$statistic, 4), ", p-value < ",
             ifelse(round(dw_test$p.value) == 0, "2.2e-16",
                    dw_test$p.value), "\n"))
  cat(paste0("alternative hypotesis: ", dw_test$alternative, "\n"))
  invisible()
}
```

RNN_predict

```
RNN_predict <- function(object, sigb, epsp){
  predict <- .SM.predictRNN(object, sigb, epsp)

  predict
}

.SM.predictRNN <- function(object, sigb, epsp){
  preprocess_param <- object["preprocess_parameter"]

  if( length(sigb) != length(epsp) ) stop("Data lengths must be equal.")
```

```
  X <- as.matrix(cbind(sigb, epsp))

  res <- .SM.buildRNN.preprocess(X = X, method = preprocess_param$method,
                                 pre_param_x = preprocess_param$X,
                                 pre_param_y = preprocess_param$Y,
                                 lag = preprocess_param$lag)
  new_model <- load_model_hdf5(object["model_path"])

  predict <- new_model %>%
    predict(res$X_data)

  predict
}
```

Apéndice B: Manual de ayuda

Package 'sm4sd'

May 17, 2020

Type Package

Title Statistical Modeling for Sensor Data

Version 0.1.0

Author Pan Ye

Maintainer Pan Ye <yepan0720@gmail.com>

Description Apply statistical model to analyze electrical conductivity of pore water.

License GPL (¿=2.0)

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Depends magrittr

Imports dlm,
generics,
ggplot2,
gridExtra,
lmtest,
methods,
stringr,
timeSeries,
keras

Collate '1_imports.R'
'2_package_description.R'
'3_classes.R'
'internal_predict.R'
'function-RNN_predict.R'
'internal_buildClass.R'
'internal_buildRNN.R'
'internal_extractMeasures.R'
'internal_fit.R'
'internal_function.R'
'internal_get.R'
'internal_getMod.R'
'internal_getPred.R'
'internal_plot.R'
'internal_plotCor.R'
'internal_print.R'

'internal_residualDiag.R'
'internal_show.R'
'internal_summary.R'
'methods-[.R'
'methods-buildClass.R'
'methods-buildRNN.R'
'methods-extractMeasures.R'
'methods-fit.R'
'methods-getMod.R'
'methods-getPred.R'
'methods-plot.R'
'methods-plotCor.R'
'methods-print.R'
'methods-residualDiag.R'
'methods-show.R'
'methods-summary.R'

R topics documented:

`buildClass` *Build Extended S4 Class*

Description

`buildClass` extends the class `RS` to `SM.dlm` and `SM.HL` by adding the additional informations.

Usage

```
buildClass(object, method, freq, ind, lagMax, verify, parallel)

## S4 method for signature 'RS,character'
buildClass(object, method, freq, ind, lagMax,
  verify, parallel)
```

Arguments

object	An object of RS class.
method	An character value indicating the method to apply. Available method in this package are: • dlm: build a SM.dlm class; • hl: build a SM.HL class.
freq	Object with type ANY. • dlm method: a positive intervalue indicating the seasonality of the time series' structure; • hl method: a NULL value, it is not necessary for SM.HL class.
ind	R object with type ANY. • dlm method: a list containing outliers or human intervention indexes; • hl method: a numeric value indicating the offset value.
lagMax	R object with type ANY. For dlm method, lagMax is an integer value indicating the maximum effect lag for each outlier. Check section Details in SM.dlm for more information.
verify	R object with type ANY. For dlm method, verify is a logical value indicating whether the validation process should be applied. Check section Details in SM.dlm for more information.
parallel	R object with type ANY. For dlm method, verify is a logical value indicating whether the parallel process should be applied.

Examples

```
n <- 50
# An example to create the RS class.
my_first_class <- RS(sigb = rnorm(n, mean = 0, sd = 1),
                     epsb = rnorm(n, mean = 0, sd = 2),
                     epsp = rnorm(n, mean = 1, sd = 1))

# build class
## build dlm model
dlm_class <- buildClass(object = my_first_class,
                        method = "dlm", freq = 1,
                        ind = list(), lagMax = 3,
                        verify = TRUE, parallel = TRUE)
## build HL model
hl_class <- buildClass(object = my_first_class,
                       method = "HL", ind = 9.5)
### ind in this case is offset
```

buildRNN *Build RNN model*

Description

buildRNN extends the class RS to SM.rnn by constructing and validating Keras models.

Usage

```
buildRNN(object, compile_options, hyperparameters, preprocess_method, lag,
  model_path)

## S4 method for signature 'RS,list,list,character,numeric,character'
buildRNN(object,
  compile_options, hyperparameters, preprocess_method = c("normalized",
  "maxmin"), lag, model_path)
```

Arguments

object	A RS class.
compile_options	A list contaning the follow compile options: optimizer, loss and metrics. In case of missing ones of those indicators, default values are setted automatically: • optimizer: adam • loss: mean_absolute_percentage_error • metrics: mean_absolute_percentage_error . For more detail, check compile function of Keras.
hyperparameters	A list contaning the follow parameters of Keras model: • batch_size: Integer or NULL. Number of samples per gradient update. If unspecified, batch_size will default to 32, • epochs: Number of epochs to train the model. If unspecified, epochs will default to 10, • test_split: the percentage of trained data that goes to testing. Default value 0.3, • validation_split: the percentage of data that goes to validading model. Default value 0.3. Note: a vector input is acceptable for each hyperparameter, in this case gridsearch will be applied.
preprocess_method	A character value indicating data preparation method, available methos are • normalized: normalization method using scale function, • maxmin: maximum minimum transformation.
lag	A integer value indicating how many previous data should be included in the data preparation process.
model_path	A string value indicating where should the model to be stored.

Detail

If lag is diferent a zero, the input data will be transformed to a new input with column number equal to 2x(lag + 1), named n_col. Once the data preparetion is done including the preprocess method, a Keras sequential model is constructed with following layers:

- a simple layer with input shape equal to c(n_col, 1) and output shape equal to n_col,
- a simple rnn layer with n_col units,
- a simple layer with one unit with linear activation.

This model will be trained by first (1 - validation_split)% rows (the last test_split part to test the model) and validated by the last validations_split%. Once the model is done, it will be store to the model path because the S4 object can not treat the keras model as slot object.

Examples

```
## Not run:
n <- 50
# An example to create the RS class.
 my_first_class <- RS(sigb = rnorm(n, mean = 0, sd = 1),
                      epsb = rnorm(n, mean = 0, sd = 2),
                      epsp = rnorm(n, mean = 1, sd = 1))
 compile_options <- list()
 hyperparameters <- list(batch_size = c(16, 32, 64),
 epochs = c(10, 20, 50))
 preprocess_method <- "maxmin"
 lag <- 5
 model_path <- "~/Desktop/pans_model"
myRNN <- buildRNN(object = myclass, compile_options = list(), hyperparameters = hyperparameters,
 preprocess_method = preprocess_method, lag = lag, model_path = model_path)

## End(Not run)
```

---

extractMeasures *Extract and Plot Measurement*

Description

Extract and plot the electrical conductivity of pore water.

Usage

```
extractMeasures(object, plot)

## S4 method for signature 'SM.dlm.fitted,logical'
extractMeasures(object, plot)

## S4 method for signature 'SM.HL,logical'
extractMeasures(object, plot)
```

Arguments

object	Object to be extracted the stored measurement.
plot	A logical value indicating whether the graphic should be plotted.

Note

Check the section Examples in SM.dlm.fitted and SM.HL for practical use in each case.

---

fit *Build and train the SM.dlm Class in Package sm4sd*

Description

Creating an object of SM.dlm.fitted class by build and train the dlm model based on the information of input.

Usage

```
fit(object)

## S4 method for signature 'SM.dlm'
fit(object)
```

Arguments

object	An object of SM.dlm.

Note

Check the section Examples in SM.dlm.fitted for practical use.

---

getMod *Extract and/or Build the Corresponding Model*

Description

Rebuilding the model based on the informations stored in the object.

Usage

```
getMod(object, name)

## S4 method for signature 'SM.dlm.fitted,missing'
getMod(object)
```

Arguments

object	An existing class in this package.
name	Missing value, it's not defined for current version.

Note

Check the section Examples in SM.dlm.fitted for practical use.

plot *Methods for Function Plot in Package sm4sd*

Description

Plot slots of RS class using "timeSeries" plot.

Usage

```
## S4 method for signature 'RS,missing'
plot(x, y, ...)
```

Arguments

x	An object of RS class.
y	Missing object.
...	Additional graphical arguments, see plot, plot.default and par.

Examples

```
n <- 50
# An example to create the RS class.
my_first_class <- RS(sigb = rnorm(n, mean = 0, sd = 1),
                     epsb = rnorm(n, mean = 0, sd = 2),
                     epsp = rnorm(n, mean = 1, sd = 1))

# plot the class as timeseries object
plot(my_first_class)
```

plotCor *Correlation Plot*

Description

Correlation plot for the x_t (product of ϵ_b, σ_p) and y_t (σ_b) by groups.

Usage

```
plotCor(x, groups, ...)

## S4 method for signature 'RS,numeric'
plotCor(x, groups, ...)
```

Arguments

x	An object of RS class.
groups	A positive integer indicating the number of subsets with equal length and preserving the structure order. Valid values: • 1: unic linear regresion; • 2 - 6: linear regresion by group depending the number of group; • bigger than 6: exceeded group, pass to 6 groups.
...	Additional graphical arguments, see plot, plot.default and par.

Examples

```
n <- 50
# An example to create the RS class.
my_first_class <- RS(sigb = rnorm(n, mean = 0, sd = 1),
                     epsb = rnorm(n, mean = 0, sd = 2),
                     epsp = rnorm(n, mean = 1, sd = 1))
# plot the correlation
plotCor(my_first_class, 7)
```

---

print *Print Values*

Description

print class values.

Usage

```
## S4 method for signature 'RS'
print(x, ...)

## S4 method for signature 'SM.dlm'
print(x, ...)

## S4 method for signature 'SM.dlm.fitted'
print(x, ...)

## S4 method for signature 'SM.HL'
print(x, ...)

## S4 method for signature 'SM.rnn'
print(x, ...)
```

Arguments

x	An S4 class.
...	Additional argument for print where is not used in this pakcage.

residualDiag *Residual Diagnostic for DLM Model*

Description

Residual Diagnostic including several normal test and residual plots.

Usage

```
residualDiag(object, gamma)

## S4 method for signature 'SM.dlm.fitted,numeric'
residualDiag(object, gamma)
```

Arguments

object	An object to be analyzed.
gamma	Numeric value between 0 and 1 indicating the confidence level.

Details

The residual diagnostic include the confidence band plot, chisq test for normalized error plot, Quantile-Quantile Plots and Durbin-Waston Test for autocorrelation.

RNN_predict *Predict Function for SM.rnn Class*

Description

Simulation of ϵ_b based on the new ϵ_p and σ_b according to the pattern captured by SM.rnn model.

Usage

```
RNN_predict(object, sigb, epsp)
```

Arguments

object	An SM.rnn class.
sigb, epsp	A n-dimensional numeric vector of electrical conductivity of bulk soil measures and electrical permitivity of pore water measures.

Note

Before the prediction is applied, the data will be preprocessed according to the preprocessed method defined in the SM.rnn class.

RS-class *Class RS — An S4 class to represent a raw signal inputs*

Description

The RS class contains 3 equal length vector inputs: σ_b, ϵ_b and ϵ_p.

Slots

`sigb` A n-dimensional numeric vector represents the electrical conductivity of bulk soil, σ_b.

`epsb` A n-dimensional numeric vector represents the electrical permitivity of bulk soil, ϵ_b.

`epsp` A n-dimensional numeric vector represents the electrical conductivity of pore water, ϵ_p.

Details

Before the class is created, `setValidity` will be launched to check the length of 3 input vectors. It will return "Error" if the lengths are not equal.

Methods

Available methods for this class are:

- `show`: the class display is simplyfied, each slot is limited to 25;
- `print`: displays the whole class;
- `get` ('['): returns the value of a slot;
- `plot`: plots 3 slots as "timeseries" objects in the same graphic;
- `plotCor`: plots the correlation between ϵ_b and σ_b;
- `buildClass`: builds classes `SM.dlm` and `SM.HL`;
- `buildRNN`: training a RNN model in order to build clase `SM.rnn`.

Check the example below for more details.

Examples

```
n <- 50
# An example to create the RS class.
my_first_class <- RS(sigb = rnorm(n, mean = 0, sd = 1),
                     epsb = rnorm(n, mean = 0, sd = 2),
                     epsp = rnorm(n, mean = 1, sd = 1))

# Show the class
my_first_class
# print the class
print(my_first_class)
# Extract the slot "sigb"
sigb <- my_first_class["sigb"]
### o  sigb <- `[`(my_first_class, "sigb)

# plot the class as timeseries object
plot(my_first_class)
```

```
# plot the correlation
plotCor(my_first_class, 7)

# build class
## build dlm model
dlm_class <- buildClass(object = my_first_class,
                        method = "dlm", freq = 1,
                        ind = list(), lagMax = 3,
                        verify = TRUE, parallel = TRUE)
## build HL model
mp_class <- buildClass(object = my_first_class,
                       method = "HL", ind = 9.5)
### ind in this case is offset
```

---

show *Show an Object*

Description

Display the object by printing according to the class object. Check "Details" section for more info in each case.

Usage

```
## S4 method for signature 'RS'
show(object)

## S4 method for signature 'SM.dlm'
show(object)

## S4 method for signature 'SM.dlm.fitted'
show(object)

## S4 method for signature 'SM.HL'
show(object)

## S4 method for signature 'SM.rnn'
show(object)
```

Arguments

object	An s4 class.

Details

Simplying the print for each case:

- RS: first 25 elements for each slot;
- SM.dlm: call, seasonality of time series, index of event indicators, extra parameter as lagMax, verify and parallel, and inherited part of RS class.

- SM.dlm.fitted: call, best model parameters including dlm parameter (V, W, outlier ajustment), and associated lag, seasonality, index of event indicators and first 5 rows of tracking ordered decreasely by performance.
- SM.HL: call, corresponding offset and inherited part of RS class.

SM-class *Class SM — An virtual S4 class to represent the raw signal model.*

Description

Class "SM" is a virtual super class and cornstone of specific model.

Slots

sigb,epsb,epsp Inherited slots from RS.

Details

Available model extensions in this packages are:

- SM.dlm: dynamic linear model, it's a space-state model to estimate the offset;
- SM.HL: Hilhorst model, it's a deterministic model where the offset is prefixed.

Extends

From class RS directly without extra slots.

SM.dlm-class *Class SM.dlm — An S4 class to represent the unfiited dynamic linear model for raw signal*

Description

SM.dlm class contrains the model defition based on the timeseries structure.

objects from the Class

Objects are created by the method buildClass or by calls of the form SM.dlm(...). However, the first form is recommended.

Slots

call An object of class call returning an unevaluated function call.

freq Positive integer value indicating the stationality of the raw inputs. Set it to 1 if the inputs has no stationality.

indicators A list containing outliers or human intervention indexes, see examples.

lagMax An integer value indicating the maximum effect lag for each outlier, see Details.

verify A logical value indicating whether the validation process should be applied, see Details.

parallel A logical value indicating whether the parallel process should be applied.

Details

Usually, outliers are happened in the data due to uncontrollable event or human intervention. In those cases, one should annotate the indexes where it presents to validate the model. check the example for more application detail.

Once outlier happended, its effect could be instant or remains some time period in case of timeseries. If lagMax is bigger than 1, then gridsearch will be applied: for each outlier index and for each possible lag between one and lagMax, the maximun likelihood estimation will be calculated for each possible combination. Then, best model will be returned after the gridsearch process which has the best MLE performance.

The number of models is determinated by follow equation:

$$numberofmodels = lagMax * *numberofindicators.$$

According to the formula, the gridsearch complexitity is an exponential function respect a #indicators with base lagMax. To avoid computation problem, it's best to assign verify equal to TRUE, where it returns FALSE if #indicators ** lagMax ¿= 200 in the class definition. To skip this validation process, kindly set verify to FALSE.

Methods

Available methods for this class are:

- `show`: the class display is simplyfied, each data input is limited to 25;
- `print`: displays the whole class;
- `get` ('['): returns the value of a slot;
- `fit`: training the model and returning the `SM.dlm.fitted`.
- `extractMeasures`: returning error, because the model hasn't trained.

Extends

From class `SM-class`, directly.

Examples

```
n <- 50
rs.class <- RS(sigb = rnorm(n, mean = 0, sd = 1),
               epsb = rnorm(n, mean = 0, sd = 2),
               epsp = rnorm(n, mean = 1, sd = 1))
# The inputs we created are random variable,
# then they have non-stationality structure.
freq <- 1
# Suppose there is a rain at index 1 and 20, and irrigation at 5.
index <- list(rain = c(1, 20), watering = c(5))
lagMax <- 10
verify <- parallel <- TRUE
## Not run:
# it returns error because 10 ** 3 = 1000 which is bigger than 200.
dlm.class <- buildClass(object = rs.class, method = "dlm",
                        freq = freq, ind = index, lagMax = lagMax,
                        verify = verify, parallel = parallel)

## End(Not run)
lagMax <- 3
```

```
dlm.class <- buildClass(object = rs.class, method = "dlm",
                        freq = freq, ind = index, lagMax = lagMax,
                        verify = verify, parallel = parallel)
dlm.class
```

`SM.dlm.fitted-class` *Class SM.dlm.fitted — An S4 class to represent the fitted dynamic linear model*

Description

The SM.dlm.fitted class store information about fitted dynamic linear model and its predefinition.

Objects from the Class

SM.dlm.fitted is created by applying method `fit` to the `SM.dlm` class.

Slots

`parameters` A list containing the best best parameters which are used to build the corresponded model by `getMod`.

`filtered` A list contaning the filtered value. For more information, check the Value section of `dlmFilter`.

`smoothed` A list contaning the smoothed value. See also `dlmSmooth`.

`lags` A list contaning the best lag for each outlier indicator. For more detail, see the section Details of `SM.dlm`.

`seasonalSign` A character indicating the significance of sesonal part in the model:

- null: if the data doesn't have sesonal part (freq == 1)
- TRUE: the seasonality is significant in the model
- FALSE: the seasonality is not significant

. Check details for class building.

`tracking` A data frame contaning the tracking of the gridsearch history ordered by the negative log likelihood.

Methods

Available methods for this class are:

- `show`
- `print`
- `get` ('['): giving the value of a slot
- `getMod`: returning the dlm model
- `residualDiag`: residual diagnostic for model validation
- `extractMeasures`: measurement extraction and visualization.

Extends

From `SM.dlm`, directly.

Examples

```
n <- 50
rs.class <- RS(sigb = rnorm(n, mean = 0, sd = 1),
               epsb = rnorm(n, mean = 0, sd = 2),
               epsp = rnorm(n, mean = 1, sd = 1))
# The inputs we created are random variable,
# then they have non-stationality structure.
freq <- 1
# Suppose there is a rain at index 1 and 20, and irrigation at 5.
index <- list(rain = c(1, 20), watering = c(5))
lagMax <- 10
verify <- parallel <- TRUE
lagMax <- 1

 dlm.class <- buildClass(object = rs.class, method = "dlm",
                         freq = freq, ind = index, lagMax = lagMax,
                         verify = verify, parallel = parallel)
 dlm.fitted.class <- fit(dlm.class)
 dlm_model <- getMod(dlm.fitted.class)
```

SM.HL-class *Class SM.HL — An S4 class to represent the Hilhorst model.*

Description

Class SM.HL — An S4 class to represent the Hilhorst model.

Slots

`call` An object of class `call` returning an unevaluated function call.

`offset` A no negative numeric value indicating $\epsilon_{\sigma_b} = 0$.

Note

A hilhorst model is a deterministic model for this problem. For more detail, please check `sm4sd Package` and see also the paper of M.A.Hilhorst: A Pore Water Conductivity Sensor.

SM.rnn-class *Class SM.rnn — An S4 class contains a recurrent neural network model for σ_b simulation*

Description

SM.rnn class contains the trained rnn model and its parameters.

objects from the Class

Objects are created by the method `buildRNN`.

Slots

`call` An object of class `call` returning an unevaluated function call.

`model_path` The directory where the trained rnn model should be stored.

`compile_options` A list contanning compile options of keras model, check `buildRNN` for more detail.

`optim_hyperparameters` A list contanning tunable hyperparameters, check `buildRNN`.

`preprocess_parameter` A character value indicating data preparation method, `buildRNN`.

`tracking` A data frame containing the tracking of the gridsearch history of keras model.

Note

Check the Detail section of `buildRNN` to understand how SM.rnn model is created.

Functions

Available functions for this class are:

- `show`: display the information of the RNN model
- `RNN_predict`: predict function.

`sm4sd Package` *sm4sd: A package for computing the conductivity of the pore water of soil.*

Description

The sm4sd provides several methods to extract the pore water electrical conductivity based on data including Hilhorst model (deterministic model) and time-varying dynamic linear model approach.

Details

Salt concentration of bulk is a strong indicator for world's agricultural productivity. One way to mesure it is through determining the pore water conductivity of soil, σ_p. According to Hilhorst, σ_p can be determined from the equation:

$$\epsilon_p = \frac{\sigma_p * \epsilon_b}{(\sigma_b - \sigma_{\epsilon_b=0})}$$

where

- ϵ : electrical permittivity,
- σ : electrical conductivity,
- p : pore water,
- b : bulk soil.

The values ϵ_b and σ_b can be mesured in the bulk soil using a dielectric sensor; ϵ_p is a function of temperature; and $\epsilon_{\sigma_b} = 0$ appear as a offset of the linear relationship between ϵ_b and σ_b depending on soil type. In his work, he recommended using 4.1 as a generic offset.

Once the offset is fixed, this model is deterministic. The producer of capacitance soil moisture sensors 5TE recommends the use the an offset $\epsilon_{\sigma_b} = 0$ of 6 while another study found that this value is appropriate and does not present a good linear relationship between ϵ_b and σ_b Giving the randomness, Basem, Jose and Gerd described a statistical approach using the time-varying dynamic linear model (dlm). In this work, they considered the offset $\epsilon_{\sigma_b} = 0$ as an unknow parameter which is also estimaded by the model. The model can be formulated as follow:

$$y_t = A_t x_t + v_t$$

$$x_t = x_{t-1} + w_t$$

where

- y_t is an 1-dimensional vector, representing the observation at time t, ϵ_b;
- x_t is an 2-dimensional uncorrelated vector, representing the state at time t, $(x_1, x_2)_t$, and
  - x_1 is the offset $\epsilon_{\sigma_b=0}$,
  - x_2 is related to the product of σ_b, ϵ_p;
- A_t, for our specific model, is 2-dimension vector $(1, \sigma_b \times \epsilon_p)_t$;
- v_t, w_t are white noise random variable with their own covariance matrix.

The sm4sd contrains several classes:

- `RS`: raw signal class with 3 raw inputs, σ_b, ϵ_b and ϵ_p;
- `SM-class`: signal model class, it contains the `RS` and it's virtual, it means you can't create directly an instance with this class;
- `SM.dlm`: dynamic linear model for raw signal, it contains `SM-class` and the extra parameters for the model definition;
- `SM.dlm.fitted`: fitted dynamic linear model;
- `SM.HL`: Hilhorst model;
- `SM.rnn`: Recurrent Neural Network model for σ_b simulation.

For more details, check the class definitions.

[*Extract Slots of an S4 Class*

Description

Operators acting on existing classes to extract slot's values.

Usage

```
## S4 method for signature 'RS,character,missing,missing'
x[i, j, drop]

## S4 method for signature 'SM.dlm,character,missing,missing'
x[i, j, drop]

## S4 method for signature 'SM.dlm.fitted,character,missing,missing'
x[i, j, drop]

## S4 method for signature 'SM.HL,character,missing,missing'
x[i, j, drop]

## S4 method for signature 'SM.rnn,character,missing,missing'
x[i, j, drop]
```

Arguments

x	An S4 object from which to extact slot's values.
i	A character value indicating the slot's name.
j	Missing value which is not used for objects from this package.
drop	Same as j.

Examples

```
n <- 50
# An example to create the RS class.
my_first_class <- RS(sigb = rnorm(n, mean = 0, sd = 1),
                     epsb = rnorm(n, mean = 0, sd = 2),
                     epsp = rnorm(n, mean = 1, sd = 1))
# Extract the slot "sigb"
sigb <- my_first_class["sigb"]
```

Index